The USENIX Association Magazine

;login:

DECEMBER 1997

volume 22 • number 6

inside:

features:

USENIX The Advanced Computing Systems Association

upcoming events

7th USENIX Security Symposium

Sponsored by USENIX in cooperation with the CERT Coordination Center

WHEN	WHERE	WHO
January 26-29/98	San Antonio, TX	**Avi Rubin,** Program Chair AT&T Labs - Research **Greg Rose,** Invited Talks Coord Qualcomm Australia

4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)

WHEN	WHERE	WHO *program chair*
April 27-30/98	Santa Fe, NM	**Joe Sventek** Hewlett-Packard

DEADLINES

Notification to Authors	Final Papers
January 7/**98**	March 17/**98**

System Administration, Networking, & Security (SANS) Conference

Sponsored by the SANS Institute, co-sponsored by SAGE and USENIX

WHEN	WHERE
May 9-15/98	Monterey, CA

USENIX Annual Technical Conference

WHEN	WHERE	WHO
June 15-19/98	New Orleans, LA	**Fred Douglis,** Program Chair AT&T Labs **Clem Cole,** Inv. Talks Co-coord. Digital Equipment Corporation **Barry Kercheval,** IT Co-coord. Xerox PARC **Jon "maddog" Hall,** Freenix Track Coord. Digital Equpment Corporation

DEADLINES

Notification to Authors	Full Papers	Final Papers
January 26/**98**	March 30/**98**	April 27/**98**

2nd USENIX Windows NT Symposium

WHEN	WHERE	WHO *program chair*
August 3-5/98	Seattle, WA	**Thorsten von Eicken** Cornell University **Susan Owicki** Intertrust, Inc.

DEADLINES

Paper Submissions	Break-Out Proposals	Notifications of Acceptance	Final Papers
March 3/**98**	March 20/**98**	April 3/**98**	June 18/**98**

Large Installation System Administration of Windows NT Conference

Co-sponsored by USENIX and SAGE

WHEN	WHERE	WHO *program co-chairs*
August 5-7/98	Seattle, WA	**Remy Evard** Argonne National Laboratory **Ian Reddy** Simon Fraser University

DEADLINES

Paper Submissions	All Other Submissions	Notification to Authors	Final Papers
March 3/**98**	March 10/**98**	March 31/**98**	June 18/**98**

3rd USENIX Workshop on Electronic Commerce

WHEN	WHERE	WHO *program chair*
August 31-Sept. 3/98	Boston, MA	**Bennet Yee** Univ. of California, San Diego

DEADLINES

Extended Abstracts	Notifications to Authors	Final Papers
March 6/**98**	April 17/**98**	July 21/**98**

6th Annual Tcl/Tk Conference

WHEN	WHERE	WHO *program co-chairs*
September 14-18/98	San Diego, CA	**Don Libes** NIST **Michael J. McLennan** Bell Labs

DEADLINES

Paper Submissions	Notification to Authors	Final Papers
April 8/**98**	May 11/**98**	July 28/**98**

12th Systems Administration Conference (LISA '98)

Co-sponsored by USENIX and SAGE

WHEN	WHERE	WHO
December 6-11/98	Boston, MA	**Xev Gitler,** Program Co-chair Lehman Brothers **Rob Kolstad,** Program Co-chair BSDI, Inc. **Phil Scarr,** Invited Talks Co-coord. Netscape **Pat Wilson,** Invited Talks Co-coord. Dartmouth College

3rd Symposium on Operating Systems Design and Implementation

Co-sponsored by ACM SIGOPS

WHEN	WHERE	WHO *program co-chairs*
February 22-25/99	New Orleans, LA	**Margo Seltzer,** Harvard University **Paul Leach,** Microsoft

DEADLINES

Paper Submissions	Notification to Authors	Final Papers
July 28/**98**	October 13/**98**	January 6/**99**

contents

;login:

;login: is the official magazine of the USENIX Association.

;login: (ISSN 1044-6397) Volume 22, Number 6 (December 1997) is published bimonthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$30 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $50 per year.

Periodicals postage paid at Berkeley, CA and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

Editorial Staff

Editor:
Rob Kolstad <kolstad@usenix.org>

SAGE News and Features Editor:
Tina Darmohray <tmd@usenix.org>

Standards Report Editor:
Nick Stoughton <nick@usenix.org>

Managing Editor:
Eileen Cohen <cohen@usenix.org>

Copy Editor:
Sylvia Stein Wright

Proofreader:
Kay Keppler

Designer:
Vinje Design

Typesetter:
Festina Lente

Advertising

Cynthia Deno <cynthia@usenix.org>

Membership and Publications

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 510 528 8649
FAX: 510 548 5738
Email: <office@usenix.org>
WWW: <http://www.usenix.org>

The closing dates for submission to the next two issues of *;login:* are February 3, 1998 and April 7, 1998.

in this issue . . .

by Ellie Young
Executive Director
<ellie@usenix.org>

As I write this, we've just returned from a highly successful LISA '97 conference in San Diego (you shoulda been there!), so the juxtaposition of Hal Miller's SAGE column and Elizabeth Zwicky's always provocative views makes for particularly topical reading.

In fact, this is an opinionated issue: Lee Damon, across the way, thinks the Web is a fad; Tina Darmohray thinks pagers are good for you; Phil Cox isn't so skeptical about Microsoft anymore; an anonymous contributor gives a backhanded compliment to NT; Rik Farrow keeps tearing down old fences; Dave Taylor writes poetry; and Peter Salus . . . well, you know Peter.

Actually, we thrive on opinions, strongly expressed. This magazine exists in great part for you to let everyone know what you think about topics of interest to the community: use it.

We hope you enjoyed the special issue on NT you received last month. We expect to produce special issues on particular topics from time to time. If you have any ideas about a topic you would like to see covered, let us know (and volunteer to be the guest editor!).

On behalf of the USENIX Board, staff, and the *;login:* editorial team, best wishes for the holiday season!

Statement of Ownership, Management, and Circulation, 11/1/97.

Title: *;login:*. Pub. No. 008334. Frequency: Bimonthly. Six issues published annually. Subscription price: $50 individuals and institutions. Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, Alameda County, CA, 94710. Headquarters of Publication: Same. Publisher: USENIX Association, 2560 9th Street, Suite 215, Berkeley, CA 94710. Editor: Rob Kolstad. Managing Editor: Eileen Cohen, both located at office of publication. Owner: USENIX Association. The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes has not changed during the preceding 12 months.

Extent and nature of circulation	Average no. of copies of each issue in preceding 12 months	Actual no. of copies of single issue published nearest to filing
A. Total no. of copies	9347	10279
B. Paid Circulation, Mail Subs.	8577	9112
C. Total Paid circulation	8577	9112
D. Free Distribution	651	1060
E. Total Distribution	9227	10172
F. Copies not distributed	119	107
G. Total	9347	10279
H % Paid circulation	92%	89%

I certify that the statements made by me above are correct and complete.
Ellie Young, Executive Director

imho:WWWhither(ing) Internet

by Lee Damon

Lee (a.k.a. nomad) has run castle.org since 1982. He is a senior systems administrator for a San Diego-based R&D company, and is never afraid to share his opinions about computing (and he doubts that his employer would wish to claim them).

<nomad@castle.org>

Everyone's using them now. They're everywhere. Companies put them in their commercials. Kids compare them in class. Nerds (note the e) think they're k00l because they "own" one. It seems that URLs are taking over.

It's the fad of the decade: "My address is http://. . ." Web hosting companies are the Video stores of the 90s. Looking for a quick buck? Start an ISP, or better yet, just dedicate your company to Web hosting. (Who needs to deal with all those pesky modems? There's no money in that.) Anyone with a PC and some disk space can get into the act. Toss in an ISDN connection, or a frame relay line, and you're in business. BSD/OS, FreeBSD, NetBSD, Linux all have tuning to help Web services run faster. Even NT is getting into the act, and some brave folks (fools?) make the effort with win95.

Organizations like Prodigy, Netcom, AOL, Joe's ISP, and CompuServe make fortunes off "the Web." Free data they can sell to their customers! What better formula is there for getting rich? Install a link to the Internet at, oh, maybe $6,000/mo., and make $600,000/mo. Subtract a few dollars to pay for a small (overworked and understaffed) "customer support" group to answer questions like "what's a modem?" and the rest can go directly into your shareholders' pockets.

No real development costs, no paying nasty things like royalties to the people who invented the technology to make it possible. What venture-capitalist bliss.

You see URLs at the end of TV commercials, hear them in radio spots, find them splattered all over billboards and magazine/newspaper adds. Any company that's anyone has a Web page . . . or two . . . or three. Or a dozen. Want to sell your trinket? Looking for that perfect job applicant? Want to tell the world about your god(s)/goddess(es)? Plan to chase UFOs? Get a Web page and watch the hit rate climb. Not happy with your hit count? Put the word "sex" in your page, or pay Yahoo or AltaVista (or any number of other locators) and get your entry moved to the top of the list.

Lots of people have them too. At times it seems like you can't converse with anyone who doesn't have a Web page. It's self-publishing for the masses. Anyone with a message can publish on the Web. Mass communication at it's proverbial best – or worst. Just how many shrines to Elvis does this world need?

Everyone's surfin'. People spend hours, even days, just surfin' the Web. It has replaced TV in some households. The new refuge for the couch potato. Replace the remote with a mouse, and you have millions if not billions of things to stare at. Of course, the most popular pages are the ones that sell sex. At least those couch potatoes have healthy (?) libidos. (Though it gives one pause, I'll not follow that line of thought any further.)

Perhaps the saddest cut of all is that all these new, lost users keep thinking The Web *is* The Internet.

As with previous fads that made people rich, CB radio in the 70s, video stores, and object-oriented code in the 80s, this one will eventually saturate. People will tire of looking at the latest pictures of their neighbor's newborn, or reading about Uncle Fred's Caribbean vacation, or hearing Joe's latest story about cat fungus. They'll put the computer aside, switch off the modem, and move on to the next thing to grab the public's fancy.

Maybe the modem will be turned back on when Jimmy needs to do some research on his paper about dinosaurs, or Sally wants to work on her biology report, or dad wants to find a new recipe for grilled chicken, or mom wants to send email to her sister in Seattle. Generally the average household computer will once again be relegated to games, a bit of email, and the occasional balancing of the checkbook.

A few years ago, a commentator on NPR compared the Internet to a small town with friendly neighbors where everyone knew everyone else and no one had to lock their doors. Then word got out about what a nice place it was to live, and all the tourists started showing up. Doors had to be locked, and all the friendly neighbors were smothered behind rows of ugly apartment blocks.

Right now, this small town is suffering from its own success, building everywhere, growing in leaps and bounds, without control or thought. Eventually, like No Name City in "Paint Your Wagon," it's going to collapse under its own weight. After the dust settles, it will once again be the domain of old timers and the few clued individuals who have discovered that the Internet is much more than just a Web and some Usenet posts.

I can't wait.

In the meantime, unlike the AOLs of the world, I want to thank the people who made it possible; the early ARPAnet developers, the people who worked on IP to make it viable, the people who pushed the Internet to be strong and fast, the developers whose work is so fundamental to the Internet that we've all forgotten it's there. All of us who make our livings from this industry, all of us who spend so much time at this as a hobby, anyone who's ever sent email to Aunt Sally, or received a file from Cousin Jed; we all owe these people at least a "thank you." Now if only they could collect royalties.

"imho" is an occasional column published in this space. If you feel strongly about a subject that would be of interest to our community, we will air it here. Please send your article to <login@usenix.org>

USENIX Cancels New Network Technologies Symposium

USENIX regrets that the New Network Technologies Symposium, which had been scheduled for March 2-3, 1998, has been cancelled because of an inadequate number of paper submissions.

Erratum

The previous issue of *;login:* (special issue on Windows NT, November 1997) included incorrect deadlines in the Upcoming Events calendar for the 6th Annual Tcl/Tk conference. Please see this issue's calendar and the Call for Participation for correct information.

USENIX news

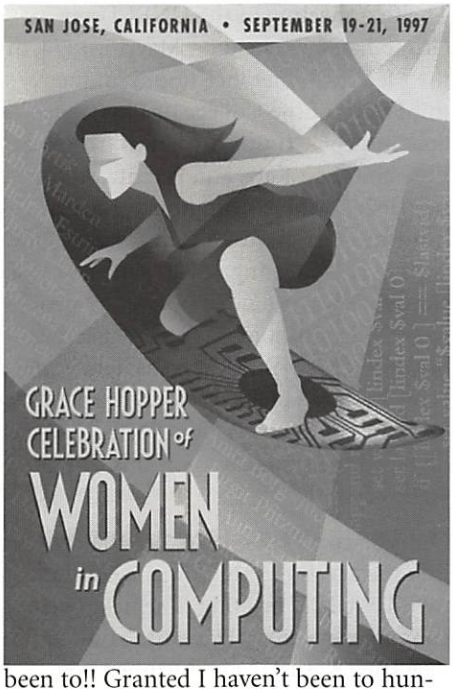
Report from a Grace Hopper Celebrant

by Nancy E. Reed

An adjunct assistant professor at the Univ. of California, Davis, Nancy is involved in research in artificial intelligence and knowledge-based systems. Her recent work has helped develop diagnostic systems that were applied to computer hardware faults and congenital heart defects.
<nereed@ucdavis.edu>

[*Editor's Note:* The Grace Hopper Celebratation of Women in Computing took place on September 19-21, 1997, in San Jose, California. USENIX was a Contributing Patron to the Conference, having donated $25,000 to supplement the scholarships and travel grants that the NSF provided for students wishing to attend the conference. Nancy Reed was one of 32 USENIX-sponsored scholarship recipients.]

The Grace Hopper conference was an exceptional experience for me. I met a number of senior women in computer science, as well as a number of students and other recent graduates. The talks were all top-notch – timely, thoughtful, clear, concise, and they really conveyed a passion for work in the field. This was also true of the more inexperienced speakers I heard, including students. A great deal of preparation was evident for all the talks. (This has not been uniformly the case at any other conference I've been to!! Granted I haven't been to hundreds, but I have been to at least six other conferences.)



I would like to commend the organizers on the food – having meals prepaid and near the meeting rooms made it possible and convenient for me to meet many more people than I would have been able to otherwise. I will miss that at future conferences I attend.

My sincere thanks to Scholarships Chair Tracy Camp and to USENIX for sponsoring my scholarship. I would not have been able to attend otherwise.

With scheduled events from 8 or 8:30 AM until 9 or 11 PM, the conference kept most of us extremely busy for over 12 hours per day. I was exhausted as well as inspired by the talks and meetings I attended. I wish the conference could be held more frequently. I will definitely try to attend again in the future.

Twenty Years Ago in ;login:

by Peter H. Salus

Peter H. Salus is the author of *A Quarter Century of UNIX* (1994) and *Casting the Net* (1995). He has known Lou Katz for over 40 years.

The November 1977 issue of *;login:* was dominated by three things: the untimely death of Joe Ossanna, the publication of K&R; and Mel Ferentz's move from Brooklyn College to Rockefeller University.

Mel wrote:

WEB SITE
http://www.usenix.org

MEMBERSHIP
Telephone: 510 528 8649
Email: <office@usenix.org>

PUBLICATIONS
Eileen Cohen
Telephone: 510 528 8649
Email: <cohen@usenix.org>

USENIX SUPPORTING MEMBERS
Adobe Systems, Inc.
Advanced Resources
ANDATACO
Apunix Computer Services
Auspex Systems, Inc.
Boeing Company
Crosswind Technologies, Inc.
Digital Equipment Corporation
Earthlink Network, Inc.
Invincible Technologies Corp.
Lucent Technologies, Bell Labs
Motorola Research & Development
MTI Technology Corporation
Nimrod AS
O'Reilly & Associates
Sun Microsystems, Inc.
Tandem Computers, Inc.
UUNET Technologies, Inc.

> The UNIX community was saddened, and greatly diminished, by the recent death of Joseph F. Ossanna. As author of NROFF/TROFF, he spoke to us at the Urbana meeting and those of us who used early versions of the typesetter package will forever be in his debt for the generosity with which he gave of his time to explain the obvious to us. We join with his many friends at Bell Laboratories in expressing our sympathy to his family.

Without Ossanna's innovative programs, we might never have Kernighan and Cherry's `eqn` (1975) or Lesk's `tbl` (1976), to say nothing of other important preprocessors.

The November 1977 *;login:* also proudly announced "On February 27, 1978, Prentice-Hall will publish *The C Programming Language* by Dennis Ritchie and Brian Kernighan ($9.95). Need we say more?" No, no more needed to be said; so Mel went on to apologize – for a failing that continued for nearly a decade more:

> We are still playing "catch up." This newsletter is being prepared in early February 1978 and is the last issue of Volume 2. The next issue, nominally December-January, will be mailed before the end of February.

There was also an apology that "no copies of the Toronto [tape] distribution have been mailed yet" and a remarkable notice: "This page appears twice in this issue." The first was the "usual photo-typesetter output," the second the "product of the Toronto typesetter simulator for the Versatec." The Toronto simulator produced incorrect intercharacter spacing because it was set for Sovran and printing in Roman.

Ferentz's announced move to Rockefeller had an important implication for the group: "Rockefeller has agreed to act as fiscal agent for the Users' Group." Previously, the finances had been handled on a more informal basis.

The first meeting of the UNIX Users Group had been held on May 15, 1974, in the Merritt Conference Room on the third floor of the College of Physicians and Surgeons. It was organized by Reidar Bornholt, Lou Katz, and Ferentz. About two dozen people from nearly a dozen institutions showed up. A year later (June 18, 1975) "a meeting at City University of New York was attended by over 40 people from 20 institutions." The first meeting outside of New York City was held at the Navy Postgraduate School, October 31, 1975, organized by Belton Allen – no one divulged the significance of this being Halloween.

On February 27-28, 1976, Berkeley got into the act: Bob Fabry organized the first two-day UNIX meeting. The next two conferences were at Harvard, "run" by Lew Law: April 1-2 and October 1-3, 1976. Steve Holmgren hosted the next meeting at the University of Illinois, Urbana, May 19-21, 1977, memorable for his barbeque and beer bash. Tom Ferrin wrote me:

> During one night at the Urbana meeting all of the attendees (about 150) went over to Steve Holmgren's house for a barbeque in the backyard. Seems to me a keg of beer was also there. This was before George Goble starting playing around with liquid oxygen for starting charcoal (see <http://ghg.ecn.purdue.edu/> for details about that), but can you imagine inviting all the attendees of a USENIX meeting over to your house nowadays? You'd have to have a place the size of the Gates' mansion.

September 12-13, 1977 saw 100 attendees at SRI in Menlo Park, California, at a meeting organized by Oliver Whitby and John Bass.

The young group was established and running: 1978 would move it further.

1998 Election for Board of Directors

by Ellie Young

<ellie@usenix.org>

The biennial election for officers and directors of the Association will be held in the Spring of 1998. A report from the Nominating Committee will be posted to comp.org.usenix and the USENIX Web site by December 15, 1997, and also published in the February issue of *;login:*. Nominations from the membership are open until January 23, 1998. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing (or five separate nominations), to the Executive Director at the Association office, to be received by noon, PST, January 23, 1998. Please include a Candidate's Statement and photograph to be included in the ballots.

Ballots will be sent to all paid-up members on or about February 18. Members will have until March 27 to return their ballots, in the envelopes provided, to the Association office. The results of the election will be announced in comp.org.usenix, the USENIX Web site, and in the June issue of *;login:*.

The Board is made up of eight directors, four of whom are "at large." The others are the President, Vice President, Secretary, and Treasurer. The balloting is preferential; those candidates with the largest number of votes are elected. Ties in elections for Directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast.

Newly elected directors will take office at the conclusion of the first regularly scheduled meeting following the election, or on July 1st, whichever comes earlier.

SAGE news & features

[Ab]used Leashes

by Tina Darmohray

Tina Darmohray, editor of SAGE News & Features, is a consultant in the area of Internet firewalls and network connections, and frequently gives tutorials on those subjects. She was a founding member of SAGE.

<tmd@usenix.org>

I've had several interesting conversations lately that, surprisingly, revolved around pager technology. The first one took place at my investment club meeting between the members and our guest speaker, an investment fund manager. She pointed out that the cell phone market had scared investors away from pager companies, resulting in a deflated price of their stocks. We talked about whether we thought paging technology was dead. Several members of the club are also system administrators. They interjected that pagers are a different technology from cell phones and have their unique strengths. First, pagers allow you to screen your messages; you can return a page if you choose to. But once you've answered a cell phone, you're stuck. Also, pagers can send you time-critical information, whether or not you answer. For example, computers can page people when something is happening: "temperature has reached 92 degrees in machine room E." A wife can tell her husband that plans have changed: "meet us at Italianos'; the kids refused to step foot in the Indian place." Neither of these messages requires a person to answer the phone, but the information still reached the recipient in realtime.

I remember when pagers first came on the scene at my office. Some folks wore them as a kind of "badge of courage"; others made the comment, "I don't do pagers." Both these groups were addressing the concern that wearing a pager often brings: it can turn your day job into a 7x24 operation, with you covering all of those hours! Of course, no one wants that kind of scenario. I know, as a manager, I sought out information from other system administration managers on how they compensated an employee who was "on call" via a pager. I found many interesting approaches. The most common went something like this: "you will be compensated [overtime pay] for scheduled hours that you are on call. On call means that you are within an hour traveling time of the site and provide 30 minute telephone response time." Note that everything is spelled out, including what hours you're on call, which makes everyone a lot happier about wearing pagers. This kind of use of pagers helps groups provide off-hours coverage in an organized and predictable way; it can benefit both the sysadmins and their users.

Another way that my group used pagers to increase our overall sysadmin "coverage" was to enlist them in our plight to justify hiring new system administrators. Hiring justification usually involved the system administrator manager (me) putting together some report on how many tasks are getting done vs. how many tasks are being left undone and how the latter queue was increasing over time. I used to prepare the report and then, inevitably, was asked to present it to the powers that be. During those meetings, my fellow sysadmins would take turns paging the daylights out of me so that the meeting attendees would see, firsthand, that the system administrators were so busy that our pagers were getting hit every four minutes. I suppose the danger in this approach is that unsympathetic managers could choose to stock up on pager batteries, rather than employees.

Of course, I've experienced paging technology at a personal level as well. My husband is tied to a pager. When it was upgraded to an alpha-numeric variety and tied to his email, I found it to be a tremendous benefit. This way I can keep a running conversation going with him while he works (and he can't really fight back). I sometimes take devious pleasure in letting him know the goings-on back on the home front. Recently, I paged him with the news that the children had found a dead rat under the house and that it would be waiting for him upon his return. "Can't wait for you to get home, dear." Before you all side with my poor husband, I'll point out that his pager, set to vibrate, deftly switches to audible tones to warn of a low battery; it's been my experience that that happens around 2:00 am.

It's those "inopportune" pages that cause all of us to question who really benefits from pagers, the person wearing one, or the person dialing one. We often hear the analogy of being put on a leash by getting a pager. My husband contends that a pager, used correctly, can work to the benefit of the wearer. He points out that, as a manager, he can be available for questions without having to be onsite all the time, especially during times his group is working around the clock to get a product out. Sure, he's tied to a pager, but the alternative is to be tied to the office. I guess that makes sense, but it's harder to swallow when he has to leave in the middle of the school play.

Yesterday I called an old school buddy of mine, now an attorney. We had been playing telephone tag for over a week. He finally left me instructions to have his secretary page him if he was not at his desk when I called, which I did. I started the conversation by apologizing for having him paged, because the call was certainly not urgent. He responded that it really was the way he preferred to do business because it made sense. He said, "Hey look, when I get a page, if I'm busy, I don't pick up the phone. I use my pager that way. I think people who don't use their pagers as a leash aren't using the technology to their benefit. So I wasn't busy, and I'm glad you had me paged so we can finally talk."

Of course, I didn't call to talk about pagers, but once he made that comment, I had to ask him what he meant by it. He came back with a story about an unhappy colleague of his who had just spent the entire weekend working on a crunch project for a client who was still furious with her, even though she completed the project by Monday. My friend asserted that her mistake was she had failed to take advantage of pager technology. Apparently, her schedule required that she travel over the weekend, but she had taken her laptop computer so she could work on the plane, etc. However, she didn't know that while she was diligently working, her client was leaving increasingly frantic voice mail messages back at the office. My friend suggested that if she had just had a pager, she could have easily known that her top-priority client was trying to reach her and given him a quick "warm fuzzy" call. That way she could have delivered the goods and kept the client happy at the same time. The analogy that my friend used for pagers was that they are like a leash for a dog. "Without a leash, the dog stays at the office; with one, the dog still gets the work done, but now gets to go out for a walk."

We've all seen the commercials of the employee making the business call from the golf course. I think my friend is right.

Why I Am Not A Professional System Administrator

By Elizabeth Zwicky

Elizabeth is technical lead of the European Desktop Project at Silicon Graphics. She was a founding member of SAGE and is currently on the USENIX Board of Directors.

<zwicky@pterodactyl.neu.sgi.com>

The first thing we need to get straight is that I am not, in fact, a professional system administrator. You might be confused on this point; I was, until some time last month. After all, I'm clearly not an amateur system administrator – I don't think I've ever administered a system for the sheer joy of it. I also believe that my work as a system administrator is of high quality and displays those semi-tangible qualities known collectively as "professionalism" (as in, "Well, painting the tail-light red with nail polish does meet the legal requirements, but it's hardly the professional way to repair it.")

But over the years in working with SAGE, it's been clear that there's a big comprehension gulf between me and many of my colleagues, often despite mutual respect that verges on hero-worship. Early on in SAGE prehistory, it became clear that you could divide the board very neatly into people who were interested in professional issues and people who wanted to do random things to make life better for system administrators. On one side, for instance, we had the people arguing for a code of ethics and certification; on the other, we had the people arguing for local groups and Web sites with FAQs on them. Each side thought the other side's most important issues were kind of cute, but not really *important*. We made progress mostly because we found occasional areas of overlap and because the two agendas are parallel. If you have enough people, you can pursue them both without major problems.

However, we occasionally end up in actual conflict. The code of ethics, for instance, is an obvious benefit for some people, whereas I find it at best a harmless idiosyncrasy and at worst an intolerable attempt to control matters of conscience. That's not actually what I want to debate right now, but it is the deciding issue that pushed me back into discussing these matters with people.

And it was in the middle of that discussion that somebody made a key observation. It's these issues, he said, that divide professional system administrators from

people who're just in it for the money – not that he'd accuse me of just being in it for the money. I hate to shatter anybody's illusions, but I'm in it for the money. I didn't know there was anything wrong with that. As I said, I've never in my life administered a system for the sheer joy of it, and I am roughly as likely to start as I am likely to start playing football for amusement. (Although I must admit my motivations are slightly mixed, I am clearly not in it for the money as much as some people, having reduced a recruiter to a state of near-speechlessness in which he could only gasp, "You're not willing to be snowed on for a quarter million dollars a year?" Apparently, I was supposed to salivate at the sound of the cash register bell.)

Now, I think it's unethical to be a minister just for the money, or even a doctor. I also think it's unethical to do a bad job at something in order to make a fast buck. I think it's unhealthy to care more about money than anything else about your job. But I see nothing wrong with picking any branch of working with computers as an honest way of making a living, doing a good job of it, and going home at night with no particular further commitment.

Partly this is because I grew up in the academic world, where jobs you might admit to having are divided roughly into three camps. There is real work, which involves adding to the world's supply of knowledge. There is honest work, which is, well, most of the other ways of making enough money to keep body and soul together. And then there are a few callings, like ministries and the arts, which you do out of sheer love for it. Being a system administrator, like being a lawyer, secretary, accountant, truck driver, engineer, or waitress, is honest work. The distinctions between these jobs have to do with things like how much they pay, how pleasant they are, and what core competencies they demand. You can lump them together in lots of ways; for instance, you can make a distinction between blue-collar and white-collar jobs, jobs that require uniforms and ones that don't, and sedentary and active jobs.

Apparently, many other people can divide these things into professions and jobs, and most of these people who are system administrators believe that system administration really is a profession and it is important to convince everybody else of it. I am deeply unclear on this concept. It is clear to me that "profession" is a cultural construct, which is high status. Something that is "unprofessional" is *bad.* Being a professional is *good.* My grandmother thinks I am a professional because the TV ads for trade schools talk about "highly paid computer professionals." Besides, I have a college degree. My mother-in-law doesn't think I'm a professional, because I have only a bachelor's degree and because in her mind any job you can wear shorts and sandals to is not a profession.

As far as I can tell, that about sums up the state of things when it comes to defining a profession. For most people in the world, system administration already is one: you have to learn a lot of stuff, and you get paid (relatively speaking) lots of money. For the rest of the world, the key indicator for whether or not something is a profession is how much you have to suffer to enter the field, and anything that doesn't involve a long and expensive degree process is just not worth considering. The difficulty of convincing people that what you do is a profession varies according to the social status of what they do, along with its mystique. Try convincing an engineer – not a Certified Netware Engineer but a genuine we-build-dams-that-don't-fall-down (often) civil engineer, for instance – that system administration is a profession in the same sense that engineering is. It isn't going to work, but when you get depressed trying, you can always amuse yourself by trying the same argument only using any nonmathematical profession (dentistry, say, or being a professor of Romance languages). Or you could try to convince a doctor that civil engineer-

SAGE MEMBERSHIP

<office@usenix.org>

SAGE ONLINE SERVICES

Email server: <majordomo@usenix.org>
Web: <http://www.usenix.org/sage/>

SAGE SUPPORTING MEMBERS

Atlantic Systems Group
Digital Equipment Corporation
Enterprise Systems Management Corporation
GNAC, Inc.
Great Circle Associates
OnLine Staffing
Pencom Systems Administration/PSA
Sprint Paranet
Taos Mountain
Texas Instruments, Inc.
TransQuest Technologies, Inc.
UNIX Guru Universe

ing is a profession in the same sense that medicine is. As a party game, this is kind of amusing, but I have better things to do in real life.

So, it turns out, I am not a professional system administrator. I am still trying to decide what I am – a practical system administrator, perhaps.

This opens up the question of what I am doing involved with SAGE, which is "dedicated to the advancement and recognition of system administration as a profession." (It seemed to me that I had seen USENIX describe itself as an association for advanced computing professionals, but we appear to currently settle for "USENIX is the advanced computing association," without specifying who or what associates.) It certainly seems to call into question how I managed to get an award for advancing the profession.

And the answer here may be a certain amount of confusion on both sides. What I consider to be nice, straightforward ways of making system administrators' lives better often strike other people as somehow involving "system administration as a profession." For instance, I'm known to believe that system administration makes a perfectly reasonable career, not just somewhere you go while you're waiting to grow up to be a programmer. I publish about "professional" (i.e., nontechnical) issues. (The idea that a squeaky octopus is more of a professional issue than a `sendmail.cf` file strikes me as a trifle bizarre, but if you divide the world into "technical" and "professional," that's what you get. This may help explain some of my confusion about the entire question.)

I am interested in advancing system administration. I consider the question of whether or not it's a profession somewhat lower in interest than the question of how many angels can dance on the head of a pin, which has a long and honorable history. I think it's naive to believe that it would reduce the political battles, or make people more willing to hire the system administrators that they need, if system administration were widely considered a profession. (Health maintenance organizations seem to have no problem treating doctors with disrespect and understaffing.) I think it's downright offensive to believe that it's necessary to be a professional to deserve respect.

I am therefore going to wander off in my unprofessional way, doing what seems to be the right thing; I invite everybody else to do the same, even if that's making system administration a profession, as long as they promise not to assume that being a professional is an inherent good that should be obvious to everyone.

President's Column

by Hal Miller

Hal Miller is president of the SAGE STG Executive Committee.

<halm@usenix.org>

This article presents my understanding of the state of, and my proposed direction for, SAGE education and certification efforts. It may be the basis for communitywide discussion and implementation of these programs. Questions and comments may be submitted to this forum or directly to me <halm@usenix.org>.

One of the charter goals of SAGE has been to address the educational requirements of its membership and of system administrators as a group. The details of our profession have been thus far learned almost exclusively on the job, generally in a rather haphazard manner. There is no widely accepted method of determining what an individual knows. We as a guild are united (as strongly as we are ever united) behind the idea of promoting a consistent education program to ensure a minimum level of qualification.

A second, less clearly defined goal, has been to create an industrywide recognition program that provides employers an indicator of the training levels achieved by current or prospective employees. What that recognition ought to be and how it might be effected have been a murky swamp, with no consensus or agreement.

Both these goals have inspired conversation and debate, sometimes intense. Neither has resulted in implementable plans or programs to date, partly because they are "tough problems" that take a lot of work to solve and partly because the organization has been busy with formation issues.

SAGE is now in a position to address these goals. Given the apparent industry demand, and the number of vendor-specific education and certification programs now available, this has become time-critical and requires immediate application of SAGE organizational and individual resources.

I define two types of education for the purposes of this article: formal and on-the-job. I also define two types of certification: topic and comprehensive.

Formal education is generally carried out in the classroom or by correspondence. It follows an "accepted" or "standard" curriculum or course of study and employs qualified instructors who present information and monitor progress. It covers theory, background material, and methods of application to practice. On-the-job training (OJT), or informal education is usually carried out in the workplace, either managed by the student directly or mentored by a more seasoned professional. It emphasizes the practical. OJT might be accomplished in an apprenticeship program, correspondence course, or nonstructured individual training.

Comprehensive certification is what most people think of when they hear the term "certification." It is common in many professions and demonstrated by exam-

Figure 1: Professional development pyramid

ples such as a license to practice medicine, dentistry, or law, a P.E., or a journeyman's certificate for electricians. These are widely accepted, and they cover large numbers of topics. Topic certification is limited in scope to one or two topics and is acceptable only to those organizations for which the specific topics are applicable. Examples of topic certifications are those given to automobile mechanics for brakes, emission control, or air conditioning service.

Vendor-specific system administration recognition methods, such as the Certified Network Engineer from Novell, or the new Sun Microsystems certificates, are a new and relatively limited form of "certification." They are not truly "topic," because they cover multiple topics. They are not "comprehensive," because they cover things from only one viewpoint and include only topics relevant to that vendor's hardware or software. This is a trend that could result in large numbers of similar programs, each limited to a given vendor, none of which would be really applicable in the standard heterogeneous site, but would be sufficient to convince some companies to require one or more from their sysadmin candidates.

A number of building blocks must be engineered and constructed in the development of education and certification programs for system administrators. There is a natural pyramid resulting, built of a series of interdependent project areas, each of which has its own reason to exist (see Figure 1). Blocks of the higher levels require those of the lower levels as prerequisites. At the base of this pyramid is the description of our job. Next comes a definition of ethical considerations that shows something about who we are. With these, we can develop an acceptable definition of a system administrator. We then tackle the training for the job as described, for both on-the-job and classroom education. Once we have all these blocks, we are ready to debate the merits of certification, then, if we are going to proceed, to define programs for certifying sysadmins, first, on a topic-by-topic basis, then in the comprehensive arena.

We already have the foundation block in the jobs descriptions booklet. This has proven immensely successful in its own right; plus it gives us the starting point for this pyramid.

There is now a code of ethics document. Debate on what it means has not subsided, nor will it stop for some time to come. We do, though, have a base from which to proceed.

Many attempts have been made to capture in a sentence or two the definition of a system administrator. One of those attempts may eventually fit our need, but thus far the "real" definition has eluded us. Our daily jobs encompass many disparate areas.

The best available definition of a system administrator, so far, is one who, as a significant part of his or her job function, manages the operational and data integrity of networked computing systems for use by others.

We "know," as a community, what sort of OJT is required to train a working sysadmin. Documenting this and drafting a program to standardize, publish, and monitor OJT are currently under way. A Short Topics booklet will be out soon that will also define a core formal curriculum. A number of universities are already teaching a course or series of courses, all of which are acting as proving groups (as well as supplying new sysadmins).

The Executive Committee is looking at a proposal by Pat Wilson (using the working title of "merit badges") that provides guidelines for teaching/learning pieces of the job independently of other pieces. Such a proposed system now gives us a few task area descriptions/checklists, and others will trickle in as they become available. These tasks both stand on their own to teach individual subjects, and begin to aggregate to form a larger education program. The system allows us to field-test and apply the lessons learned. This proposal lends itself well to the "topic certification" role, providing the framework for a recognition program without the attendant legal and political baggage. This recognition/certification by-product allows us to develop and field-test ideas that may eventually make their way into a "real" comprehensive certification process.

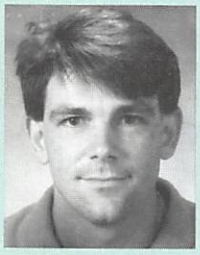
With a formal curriculum established, the OJT commonplace (thus covering both parts of the education dichotomy), and clarity with regard to the job definition, the time is right to begin the review of what certification means, what it doesn't mean, and implementation options. We will soon have practical experience, and the benefit of working programs of education and "minicertification."

System administration is not a statutorily regulated profession, as are the medical and legal fields. Those require certification in order to practice. They require adherence (including "word of honor") to a code of ethics. What we are building is different. No sysadmin will be required to do these things in order to perform the duties, use the title, or even become a member of SAGE. The purpose of these programs is to advance ourselves as individuals and as a group, not to regulate the membership.

We are at this point pursuing further clarification of the code of ethics and the codification of the merit badge proposal into a workable program. The next step is to lay out certification definitions, the pro and con arguments, and options. One of those options appears to be to set up a comprehensive plan based on various combinations of topic certifications. Results of that definition exercise will find their way to the membership soon.

We are dangerously close to being "too late" as the number of other bodies and vendors offering both education and certification increases. We must not sit idly by, but continue to press forward.

Windows NT 5.0: Integration Friendly?



by Phil Cox

Phil is a member of the Computer Incident Advisory Capability (CIAC) for the Department of Energy. He also consults and writes on issues bridging the gap between UNIX and Windows NT.

<pcc@llnl.gov>

I am on the plane home from the Microsoft Professional Developers Conference and I am amazed at the effectiveness of Microsoft. Although I am a longtime UNIX supporter, I have to admit that this conference has made me much less skeptical about Microsoft's ability to embrace good technology solutions. I left feeling that the road to decent integration between NT and UNIX is a feasible thing, and with the next release of NT (version 5.0), that process has a good start.

Although NT 4.0 has good built-in security features, the integration with UNIX-based systems has left a lot to be desired, especially in the area of secure communications. Windows NT 5.0 starts to bridge the gap in this area. With the incorporation of Active Directory (AD) – a Lightweight Directory Access Protocol (LDAP) compatible directory service – and Kerberos, a foundation for interoperability is being laid. Here is an overview of these two features and their use in intergrating NT and UNIX.

Active Directory: Microsoft's Version of NIS+

Basics

For anyone familiar with Sun's NIS+, the Active Directory will seem very familiar. The parallels between NIS and NT 4.0 domains and NIS+ and Active Directory are remarkable.

The Windows NT Active Directory provides the store for all NT domain security policy and account information. It also provides replication and availability of account information to multiple Domain Controllers [1]. The AD supports the LDAP that enables you to link the Windows NT directory with other LDAP/X500 directories. The AD also supports fine-grain access control. With this granularity, access rights can be granted down to individual properties on user objects (NIS+ {row,entry}). This enables a specific individual or group to have the right to reset passwords, but not to modify other account information. This new ability is unlike NT 4.0, which required Domain Admin permissions (i.e., total control) to modify any portion of the domain information.

Another difference in NT 5.0 is that all Domain Controllers are considered equal, so updates made on any one of them modify the Active Directory. In NT 4.0, modifications were made only to the files on the Primary Domain Controller (PDC), then propagated. The NT 5.0 structure is called "multiple master." Like NIS+, you have a PDC (master in NIS+) and as many replicas (formerly Backup Domain Controllers) as needed. Because all are considered equal, updates made on one are made and synchronized to all others automatically (just like NIS+).

Structure Changes

The structure of the directory has changed as well. Like the transition from NIS, with its flat namespace, to NIS+ and its hierarchical namespace, the Windows NT domain model changed. NT 4.0 used a flat namespace and one-way trust relationships; NT 5.0 Active Directory uses a multilevel hierarchy tree of domains. Management of trust relationships between domains in the AD is simplified through treewide transitive trust throughout the domain tree.

Usage

Unlike NT 4.0, which uses account information maintained in a secure portion of the registry on the Domain Controller, the NT 5.0 distributed security services use the Active Directory as the repository for account information. The AD improves performance, scalability, and administration. The trust relationships are also much easier to manage. The NT 4.0 model of using domain trust and pass-through authentication was much more cumbersome than the transitive trust and Kerberos delegation that comes with NT 5.0.

Security

AD security is directly dependent upon physical security and the underlying NT 5.0 OS security, primarily the security features of Access Controls and NTFS.

Kerberos: A Move Toward Integration

Windows NT 5.0 has added support for more security protocols. Currently, NT 4.0 and earlier versions utilize the Windows NT LAN Manager (NTLM) protocol. This protocol is limited to Microsoft and does not integrate in heterogeneous environments. Microsoft understood this and decided to support a protocol that is truly platform independent: Kerberos.

Kerberos Version 5 (RFC 1510) will replace NTLM as the primary security protocol for access to resources within or across Windows NT 5.0 domains. Kerberos is a standard that will provide for network authentication of heterogeneous machines. Some of the benefits of Kerberos are mutual authentication of both client and server, reduced load on the server during logon, and support for authorization delegation. Although Kerberos V5 will be the default authentication protocol, NTLM will continue [2] to be supported and used for pass-through network authentication, remote file access, and authenticated RPC connections to earlier versions of Windows NT.

NT 5.0 domains can be organized into a hierarchical domain tree. The trust relationships established between domains in the Active Directory allow users with accounts defined in one domain to be authenticated in another domain. Domain Trust relationships are established via Kerberos; thus the Kerberos transitive trust (delegation) model is in effect. This use of Kerberos will also allow the establishment of realm authentication between heterogeneous Kerberos realms.

Kerberos Highlights

Some of the NT 5.0 Kerberos highlights are:

- Each Domain Controller will implement a Kerberos Key Distribution Center (KDC).
- NT domains are equivalent to a Kerberos realm, but will still be called domains.
- The KDC uses the Active Directory as the account database for users and groups.
- Because each Domain Controller is a KDC, physical security is a *high* priority.
- NT 5.0 domains use transitive trust relationships. Thus all NT domains contained within the AD will trust each other implicitly[3].
- You can define trust relationships between existing Kerberos realms and NT 5.0 domains to generate ticket referral requests between realms and domains.
- Microsoft plans to implement extensions [4] to support public-key authentication.

Some of the benefits of Kerberos are mutual authentication of both client and server, reduced load on the server during logon, and support for authorization delegation.

Windows NT and UNIX integration is not anywhere near seamless, but it is getting better.

Heterogeneous "Realm Authentication"

This is a major win for those who want to be able to utilize single login for integrated UNIX and NT 5.0 systems (caveat: Kerberos implementations must support the Interoperability Requirements defined in RFC 1510). Although ticket referrals are supported, non-Windows NT KDCs are not likely to contain the Authorization Data NT is expecting[5]. When this occurs, Windows NT will try to use the principal name in the ticket and create a security access token for a designated user account or use a default account defined for this purpose. NT 5.0 will also integrate with DCE Security Services as well.

Authentication of External Users

NT 5.0 will provide support for public-key authentication on behalf of users who do not have a Windows NT domain account. Users will be authenticated by a public-key certificate and can be granted access to NT resources. NT 5.0 has the ability to associate one or more external users to an existing Windows NT account for access control. The subject name on the X.509 V3 certificate is used to identify the external user associated with the account. This many-to-one mapping provides a major benefit for external client integration.

Summary

Windows NT and UNIX integration is not anywhere near seamless, but it is getting better. With the addition of Active Directory and Kerberos, the job is getting easier. This is moving in the right direction. With this direction, the Windows NT 5.0 is a platform that can provide a good foundation for secure integration of heterogeneous platforms.

For more information, see <http://www.microsoft.com/ntserver> and <http://www.microsoft.com/security>.

Notes

[1] In NT 5.0, Domain Controllers are also Kerberos Key Distribution Centers. This has ramifications on trust relationships.

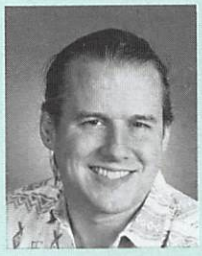
[2] Because Windows NT will continue to support NTLM authentication, you will still have to deal with the insecurities of NTLM.

[3] This is a feature of the Active Directory trust model. In NT 4.0 and earlier versions, interdomain trust relationships were established explicitly. They are defined by one-way trust relationships between Domain Controllers.

[4] A proposal to extend the Kerberos protocol specification to provide a method for using public-key cryptography for initial authentication has been submitted to the IETF working group for review.

[5] Kerberos V5 defines an encrypted field in session tickets to carry Authorization Data. NT 5.0 uses that field to hold Security IDs representing the user and group membership.

On Reliability – System Administration



by John Sellens

John Sellens has recently joined the Network Engineering group at UUNET Canada in Toronto, after 11 years as a system administrator and project leader at the University of Waterloo.

<jsellens@uunet.ca>

This time around, let's talk about system administration and how to do it more reliably. But that's what this entire series of articles is about, isn't it? Maybe I should narrow it down a little bit. Let's talk about reliability in the parts of system administration that actually takes place on a machine. First, though, let's try to define what system administration is and then discuss what things this article isn't going to be about.

"System administration" includes all the "overhead" aspects of creating and keeping a "system" running and available. Nope. This isn't working out either. Let's try another approach.

Once you have your hardware installed and your network working, there are still all the day-to-day tasks of installing software, maintaining user accounts and compiling mailing lists, monitoring, and repairing. The software- and configuration-related tasks of system administration are what I'm going to attempt to cover this time around. And I hope that's a clear enough definition, because I'm all out of ideas. For lack of a better term, I'm going to call this "system administration," but we all know that it's just one part of the total system administration workload.

Of course, no reliability article would be complete without a reminder of the basic principles: service levels, risk evaluation, costs of failures, finding the right balance, etc.

So far in this series, I've focused on hardware and wiring and touched on tasks and processes only superficially. Reliable system administration (at least as I have defined it for this article) tends to be much less related to hardware and much more related to the tasks themselves and the performance of those tasks.

I'm going to outline a few key words for reliable system administration, give a few examples of how to apply them, and then mention a few topics that don't fit neatly into those categories. (This matches nicely with system administration in general, because nothing ever fits together quite as well as you hope it will.)

Key Words

Allow me to propose a short list of important key words for reliable system administration:

- Consistency. Set your standards and procedures and stick to them; all sorts of things will be easier and more reliable.
- Documentation. If it's not properly documented, it won't pass either the "run over by a bus" test or the "won the lottery" test (i.e., if all the documentation is in your head or your mailbox, the system is unlikely to survive your sudden disappearance).
- Automation. Never do something twice by hand if you can write a program to do it for you. If tasks are (properly and carefully) automated, they're less likely to fail as a result of your being distracted or rushed or having fumbling fingers.
- Repeatability. This is often a combination of the previous three. Proper documentation and automation will help you set up a new machine (or recover or upgrade an existing one) far more effectively and reliably.

Let's talk about these in a little more detail with a few examples.

I'm convinced that one of the most effective tools in a system administrator's arsenal is consistency. Your users will thank you, your friends and family will thank you, and your evenings and weekends will thank you, too.

Consistency

This really is one of the cornerstones of system administration. If you're not consistent in how you do things, you're going to be much less effective, much less able to delegate tasks, much less able to scale your installation past a small number of systems, and much less able to put together systems that run reliably and are easily recoverable in the event of a failure.

Let's pick on everybody's favorite topic for an example: backups. (We'll be able to use this same example later for the other key words as well.) If you have inconsistent backup methods, using different commands or schedules on all your machines, I would venture to guess that you're going to find doing backups a terrible, onerous chore and not do it as well as you would hope to. And if doing backups is a complicated and convoluted task, it will be hard for you to pawn the job off on some underworked clerk (we all know, of course, that the only truly overworked people in a company are the system administrators), giving you less free time. Different backup methods for each system mean that once you get beyond a small number of machines, you will likely find it simply too complicated to do proper backups. And lastly, if you don't have one standard way of doing backups and restores, of cycling your tapes off-site, and of keeping track of which backups are on which tapes, you're going to have a heck of a time putting things back together when your company Web site goes boom, and your boss, the head of sales, the head of marketing, and the CEO are all standing around breathing down your neck.

Have I convinced you yet? OK, then how about another perennial favorite: software installation, configuration, and distribution (and if you don't believe me that it's a favorite topic, have a look at the proceedings of any of the 11 LISA conferences, and you're bound to see at least one or two related papers each year). A simple example here is the fabulous GNU "autoconf" system, which is used to generate those nice "configure" scripts that you see in so many software distributions. In the olden days, when you grabbed some software from the net, you had to read the READMEs, examine the Makefiles and .h's, and manually customize everything to suit your environment. Nowadays, if there's a "configure" script included, you can just about always rely on

```
% ./configure --prefix=/local
% make install
```

to do just what you would hope it would do, a concrete example of the time- and effort-saving benefits of consistency.

If you look back at the LISA software installation and distribution papers, one thing that they virtually all have in common is a set of rules defining how and where to install your software "packages." Decide on one place to put your source, one structure in which to install your binaries and supporting files, and follow your rules religiously. It's much easier to be able to tell someone that the source is under `/local/src/pkg` and the installation is under `/local/dist/pkg`, than to make everything you do a special case.

Consistency applies to almost everything: day-to-day procedures, how user accounts get authorized and created, where your backup tapes are stored, how IP addresses are assigned, and on and on. Whether you call them rules, policies, procedures, or practices, once you set 'em, don't forget 'em. I spent quite a few years doing centralized system administration and software support for hundreds of different machines, with different operating systems and revisions, for thousands of users; and I'm convinced that one of the most effective tools in a system administrator's arsenal is consistency.

Your users will thank you, your friends and family will thank you, and your evenings and weekends will thank you, too.

Documentation

If you don't document everything, you're not doing your job. You're making things harder for the users (is there anyone who doesn't get frustrated when the "man" page and user documentation are missing and not to be found?) and harder for yourself and your co-workers (because you'll end up answering the same questions over and over); and if all the documentation is locked up inside your head, you're making yourself unpromotable and untransferable.

There always seems to be a strong perception that "documentation is hard" or "I'm not a good writer," or "I don't have time." I'll tell you, from experience, that man pages for most programs get to be trivial to write after you've knocked off a dozen (or a gross) or so. And who hasn't had the experience of returning to a program or system after a few months and being unable to remember or determine what your beautiful, "self-documenting" code is supposed to be doing?

As I mentioned before, the favorite metric for how much documentation is enough is the "if you got hit by a bus" rule – the question being whether your systems would keep on running if you got hit by a bus tomorrow. (I prefer to use the "if you won the lottery and suddenly quit your job to move to Tahiti" rule because that seems much more cheerful.)

Make a man page (if you're a UNIX administrator) or a Web page (for anyone else) for every command or process that you install. Document each "process" or "job step," document your "cron" jobs and mail aliases, document your installation standards, your policies and procedures, your vendor service contract information, and make your life easier. Document your backup and recovery practices and processes so your minions can take care of things while you sleep. And, of course, once you have an online copy, you can make your all-important paper copy so you won't be completely stranded when your root partition dies. Set up a "cron" job or something to automatically (oops, that's the next key word) print an updated copy every few months; obsolete emergency recovery documentation isn't much fun. And finally, don't forget to write down the root password somewhere so that your systems can survive a bus crash, even if you don't (even though it's not likely to be a significant legacy handed down through the generations).

Automation

Automation is your friend. Get it working right once, on one machine, and it will (more than likely) keep right on working, and you can usually use the same method on all your other machines, old and new.

Let's look at backups again. No one wants to type the "runbackup" command every day. And it's a sure thing that no one wants to have to type nonsense like

```
% dump 3bfu 32 /dev/rmt2 / /usr /var
```

day after day after day. Get the best backup hardware and software your budget allows. If your budget is zero, at least use something like (the terrific) "amanda" backup software from the University of Maryland [1]. If your budget is larger than zero, buy expensive backup software, buy a giant jukebox, install fiber to a suitable off-site location, set it, and forget it (well, not really, but you get the idea) [2]. The idea is, of course, to make computers do the repetitive tasks – computers are good at mindless repetition, and humans generally aren't.

. . . the favorite metric for how much documentation is enough is the "if you got hit by a bus" rule – the question being whether your systems would keep on running if you got hit by a bus tomorrow.

Making tasks repeatable usually requires a larger up-front investment of time and energy.

We've all seen Makefiles (or at least, most of us have). If you think about it, `make` [3] is one of the earliest automation tools for UNIX, and it can be used for all sorts of things, in addition to building programs from source [4]. Shell scripts combined with `cron` are another easy way to automate repetitive tasks.

My final example in my argument in support of automation is its use in software distribution. One of the few recurring themes in the myriad of software distribution papers in past LISA conferences is that of automation. Updates almost always get distributed automatically, usually fired off early in the morning from a `cron` job. People can be far more effective when they have the same software environment everywhere, and, conversely, system administrators can be far less effective if they need to copy, compile, and install the same software on each machine any time something new or updated is installed.

Repeatability

There are two ways to think of repeatability – how to redo or repeat tasks on a single machine and how to apply (or repeat) the same actions across multiple machines.

Using our favorite example (backups), repeatability means that your backups can run the same way, day after day, with as little manual intervention as possible. This includes little things like making sure your log files get rolled from time to time so they don't just grow without bound and fill your disk. Repeatability also means that you can install the software and get it working again after an OS upgrade (or recovery), and it also means that once you have your backups working on one machine, you can easily get them working on all your other machines.

Making tasks repeatable usually requires a larger up-front investment of time and energy. You need to think about and write things like install scripts, Makefiles, setup scripts, and so on. In software development, we talk about the development cost being only a small part of the total cost and the bulk of the cost being in the ongoing support and maintenance of the software. I'll claim that system administration often has an analogous rule – that the bulk of the cost of system administration is often the ongoing and repetitive tasks that stretch forward through the years. But I'll also claim that the future costs can be reduced (often substantially) by making a modest investment up front in ensuring repeatability.

Two more quick examples of tasks that really benefit from repeatability: account creation (especially in an educational setting or anywhere else with high user turnover) and PC and/or workstation setup (typically through software automation, disk cloning, or custom boot disks). Very few of us look forward to creating the ten thousandth user account or setting up the fifteen hundredth "wintel" PC by hand. (Yes, repeatability is often very closely tied to automation.)

Odds and Sods

Now that we've gone over what I claim are the key words and basic ideas, I'll quickly mention a few more items that don't seem to fall obviously under one of the key words.

Use BOOTP and/or DHCP servers to dole out IP addresses and distribute DNS, gateway, and other configuration information. Even if you statically allocate IP addresses to all your PCs, X terminals, etc., the ease of setup and the ability to renumber or reconfigure easily and automatically when necessary are great ways to increase reliability (and your free time) by avoiding having to visit each desktop or having to track down which two machines are trying to use the same IP address.

Avoid manual or custom setups whenever possible. Use a standard workstation software setup, and discourage (or prevent) users from changing or adding things.

Be familiar with tools like `rdist` and `track` to automate your file distribution – they're a great way to automatically replace or repair files that have become corrupted or gone missing (as long as it's not your master host that has problems!).

Use a change control system, like RCS, SCCS, or CVS, whenever possible, and always put something meaningful in the change logs. It's a great way to keep track of what you've changed and an easy way to keep old versions available (and much more reliable than a series of `.bak` files).

Set up as much automatic monitoring as possible. Use a log file watcher to monitor syslog and other log files and dispatch warning messages in the appropriate ways (email, zephyrgrams, broadcasts, pagers, voice modems, cell phones, etc.). Or even better, have your log file watcher fire off repair scripts to fix things automatically. And you should also periodically run commands or scripts to watch things like mail and printer queues, free disk space, load average, and so on. (It's always nice to notice and repair a problem before the users notice and start calling.)

Use a `cron` job to save copies of important files (like `/etc/passwd`) to help guard against errors, accidental deletions, and filesystem corruption.

And to help you react appropriately, make sure you have the set of tools that help you do your job in the most appropriate way – tools like laptops, network connections at home, cell phones, and pagers can be annoying, but if they save you a late-night trip into work, you might be better off.

For Another Time

There are some topics that could have been included in this article but are important or large enough to merit separate discussion. In future articles, I hope to cover backups (in much deeper detail), restores, and disaster recovery and discuss how your "people practices" – training, communication, coordination, who's on call, etc. – fit into your approach to reliability, and, of course, the perennial favorite, security, and a review of how your security policies and practices affect the reliability of your systems.

Finally, if there are any reliability-related topics that I haven't covered (thereby demonstrating that I may not be as reliable a source of information as one might hope), please let me know. I'd appreciate your input.

Notes

[1] <ftp://ftp.cs.umd.edu/pub/amanda/> contains everything about Amanda, including copies of "The Amanda Network Backup Manager" by James da Silva and Ólafur Gudmundsson from LISA VII, 1993, and "Performance of a Parallel Network Backup Manager" by da Silva, Gudmundsson, and Daniel Mossé from the 1992 Summer USENIX Technical Conference.

[2] This kind of approach actually works great, by the way. Details on request.

[3] S. I. Feldman, "Make – A Program for Maintaining Computer Programs," 1978. Available at <http://plan9.bell-labs.com/7thEdMan/vol2/make>.

[4] See, for example, "'Make' as a System Administration Tool" by Bjorn Satdeva in the SANS III proceedings from 1994.

by Daniel E. Singer

Dan has been doing a mix of programming and systems administration for 13 years. He is currently a systems administrator in the Duke University Department of Computer Science in Durham, North Carolina, USA.

<des@cs.duke.edu>

ToolMan Meets PatchReport

Warning: This issue's ToolMan is highly platform specific and hard-core sysadmin. If you don't run and maintain Sun Solaris systems, you might wish to move on.

Boring Technical Stuff

OK, now that everyone else has gone away, we can get down to business. As you know, one of the very tedious aspects of maintaining computer systems is keeping them up to date via application of OS patches. This process breaks down into basically three phases: (1) identifying patches that are needed, (2) downloading patches, and (3) installing patches. You know how much fun this isn't. The process is tedious, time-consuming, error prone, and boring. But I'll show you a program that can accomplish all of this and more in one single, solitary command line.

Background

In our department, a medium-sized site, we have (among other systems) about 150 computers running Sparc and X86 Solaris (SunOS 5.5, 5.5.1, and 5.6). These all need occasional patch updates, either to fix some identifiable problems or because, well, it's just that time. Sun makes these rueful patch update tasks more bearable in a few ways. (1) They supply a nice Web site from which most patches can be downloaded – whether or not you have a support contract (though if you have the contract, you get a few extra bennies). (2) They also supply a handy program, `patchdiag`, which will analyze your system and tell you which patches could be applied. And (3) They supply some useful tools for applying and maintaining patches on your system, namely, `installpatch`, `showrev`, and, for Solaris 2.6, `patchadd`.

But there are some problems with going through this procedure. One is the use of the patchdiag program. You must connect to the SunSolve site (<http://sunsolve.sun.com> or <ftp://sunsolve.sun.com>) using your support ID and password, and download the latest copy of the `patchdiag.xref` patch database file and possibly a current copy of the `patchdiag` program. Then there are some deficiencies with the `patchdiag` output (see Listing 1).

Program: *PatchReport*
Abstract: *analyze system, download and install patches*
Platforms: *Solaris 2.x (SunOS 5.x)*
Language: *Perl 5.002+, with*
modules: libnet, Data-Dumper, MD5, and IO
Author: *Joe Shamblin <wjs@cs.duke.edu>*
Availability: *<http://www.cs.duke.edu/~wjs/pr.html>*
<ftp://x86.cs.duke.edu/pub/PatchReport/index.html>

Features are still being added, so check for updates!

Listing 1: Excerpt from the output of a run of patchdiag.

```
% patchdiag -x patchdiag.xref
================================================================================================================
System Name: turkey  SunOS Vers: 5.5.1     Arch: sparc
Cross Reference File Date: 30/Sep/97
PatchDiag Version: 1.0.1
================================================================================================================
Report Note:
Recommended patches are considered the most important and highly recommended patches that avoid the most
critical system, user, or security related bugs which have been reported and fixed to date. A patch not
listed on the recommended list does not imply that it should not be used if needed. Some patches listed
in this report may have certain platform specific or application specific dependencies and thus may not
be applicable to your system. It is important to carefully review the README file of each patch to fully
determine the applicability of any patch with your system.
================================================================================================================
INSTALLED PATCHES
Patch    Installed    Latest   Synopsis
 ID      Revision     Revision
-----    ---------    -------- ---------------------------------------------------------------------------
103582     01           15     SunOS 5.5.1:        /kernel/drv/tcp and /usr/bin/netstat patch
103630     01           09     SunOS 5.5.1:        ip ifconfig arp udp icmp patch
103663     01           08     SunOS 5.5.1:        libresolv, in.named, named-xfer, nslookup & nstest pa
103690     03           05     SunOS 5.5.1:        cron/crontab/at/atq/atrm patch
104433     03           04     SunOS 5.5.1:        pam security patch
================================================================================================================
UNINSTALLED RECOMMENDED PATCHES
Patch    Installed    Latest   Synopsis
 ID      Revision     Revision
-----    ---------    -------- ---------------------------------------------------------------------------
103558     N/A          10     SunOS 5.5.1:        admintool/launcher fixes  plus various swmtool fixes
103594     N/A          10     SunOS 5.5.1:        /usr/lib/sendmail fixes
103600     N/A          18     SunOS 5.5.1:        nfs, tlimod and rpcmod patch
103612     N/A          33     SunOS 5.5.1:        libc, libnsl, libucb, nis_cachemgr and rpc.nisd patch
103640     N/A          12     SunOS 5.5.1:        kernel patch
103680     N/A          01     SunOS 5.5.1:        nscd/nscd_nischeck rebuild for BIND 4.9.3
103686     N/A          02     SunOS 5.5.1:        rpc.nisd_resolv patch
103696     N/A          02     SunOS 5.5.1:        /sbin/su and /usr/bin/su patch
[ ... 29 lines deleted ... ]
103901     N/A          08     OpenWindows 3.5.1: Xview Patch
104338     N/A          02     OpenWindows 3.5.1: libXt patch
105251     N/A          01     OpenWindows 3.5.1: libXt Binary Compatibility Patch
================================================================================================================
UNINSTALLED SECURITY PATCHES
NOTE: This list includes the Security patches that are also Recommended
Patch    Installed    Latest   Synopsis
 ID      Revision     Revision
------   ---------    -------- ---------------------------------------------------------------------------
103558     N/A          10     SunOS 5.5.1:        admintool/launcher fixes plus various swmtool fixes
103594     N/A          10     SunOS 5.5.1:        /usr/lib/sendmail fixes
103612     N/A          33     SunOS 5.5.1:        libc, libnsl, libucb, nis_cachemgr and rpc.nisd patch
103640     N/A          12     SunOS 5.5.1:        kernel patch
103680     N/A          01     SunOS 5.5.1:        nscd/nscd_nischeck rebuild for BIND 4.9.3
103686     N/A          02     SunOS 5.5.1:        rpc.nisd_resolv patch
103696     N/A          02     SunOS 5.5.1:        /sbin/su and /usr/bin/su patch
[ ... 22 lines deleted ... ]
103901     N/A          08     OpenWindows 3.5.1: Xview Patch
104338     N/A          02     OpenWindows 3.5.1: libXt patch
105251     N/A          01     OpenWindows 3.5.1: libXt Binary Compatibility Patch
================================================================================================================
```

Got a tool that's useful, unique, way cool? ToolMan will make you famous! Please send a description of your intellectual property to <ToolMan@usenix.org>.

As you can see, the patches are listed in three sections: Installed, Recommended, and Security. This layout is not very clear in that there is overlap between the Recommended and Security sections that makes it difficult to decide which patches to install. You cannot easily see, for instance, which patches are both Recommended *and* Security without doing a lot of tedious cross-referencing. Additionally, if a patch is already installed and an update is available (Installed section), there is no indication as to whether such a patch is Recommended or Security or both. And even if these were not problems, you still have to go through the rigamarole of downloading the patches and running the installations. If you need to do this for multiple OS releases for multiple hardware architectures, it becomes easy to let this slip down to a lower spot on your priority list. This is quite unfortunate in a scenario, for example, of Internet-connected computers needing security patches!

A Break in the Clouds

One of my co-workers, Joe Shamblin, is our OS and security guru and is generally the one burdened with the chore of keeping our systems up-to-date with the latest patches. He has brilliantly automated and simplified this odious task.

Joe created – and regularly employs – a not-so-trivial Perl script called `PatchReport`, so named because in its initial incarnation its sole function was to produce a report à la `patchdiag`, but with two improvements: it automatically downloaded the `xref` file, and it provided a more usable, consolidated report. The table produced by `PatchReport` combines the information from all three sections of the `patchdiag` report, indicating in a single line for each patch if it is Recommended, Security, and/or if a prior version is already installed. This makes it much easier to scan the list and decide which patches are appropriate for your system.

But `PatchReport` has evolved into a full-fledged patch analyzer (the report), downloader, *and* installer. As noted earlier, it can accomplish all of this and leave a Solaris system completely up to date via a single command. Invoked with appropriate options specified, `PatchReport` will: connect to the Sunsolve site, download the `patchdiag.xref` and `CHECKSUMS` files, analyze patches on your system and produce a report, download selected patches, check the `md5` checksums, and install the patches. You can even tell it to shut down and reboot the system when it's done! Listing 2 shows an example of a `PatchReport` run.

By the way, if PatchReport finds that your system is completely up to date, well, you'll just have to find out for yourself what happens.

You can also use PatchReport in conjunction with Sun's JumpStart suite (used to install new systems). PatchReport can download appropriate patches to a directory. Then JumpStart can be configured to install these patches to the OS during the initial installation. Alternatively, JumpStart could be configured to call PatchReport directly.

PatchReport is written for Solaris and not extended to other operating systems because Sun has a very open and well-documented patch system that makes programs like patchdiag and PatchReport possible. Other vendors, please take note.

Closing Remarks

To paraphrase someone who was paraphrasing Dave Barry, systems administration consumes time like Dave Barry's dog gobbles taco chips. If your job involves installing, maintaining, and updating Solaris-based systems, here's a power tool that can save you hours at a pop. But you better hurry and pick up a copy before SprintNet goes down again! To paraphrase another sage, "you snooze, you looze."

Listing 2: Excerpt from the output of a run of PatchReport

```
# PatchReport -Ari
Please provide the account and password in the form "ID/passwd"
account/passwd? *******/*******
Analyzing needed patches on your machine, this might take a minute or two depending on the options you
chose, and/or your net connection.
Patch-ID    Security  Recommended  ID   Description
--------    --------  -----------  --   -----------------------------------------------------------------
103558-10   Security  Recommended       SunOS 5.5.1: admintool/launcher fixes plus various swmtool
103582-15   Security  Recommended  01   SunOS 5.5.1: /kernel/drv/tcp and /usr/bin/netstat patch
103594-10   Security  Recommended       SunOS 5.5.1: /usr/lib/sendmail fixes
103600-18   N/A       Recommended       SunOS 5.5.1: nfs, tlimod and rpcmod patch
103612-33   Security  Recommended       SunOS 5.5.1: libc, libnsl, libucb, nis_cachemgr and rpc.ni
103630-09   Security  Recommended  01   SunOS 5.5.1: ip ifconfig arp udp icmp patch
103663-08   Security  Recommended  01   SunOS 5.5.1: libresolv, in.named, named-xfer, nslookup & n...
103680-01   Security  Recommended       SunOS 5.5.1: nscd/nscd_nischeck rebuild for BIND 4.9.3
103686-02   Security  Recommended       SunOS 5.5.1: rpc.nisd_resolv patch
103690-05   Security  Recommended  03   SunOS 5.5.1: cron/crontab/at/atq/atrm patch
[ ... 21 lines deleted ... ]
103901-08   Security  Recommended       OpenWindows 3.5.1: Xview Patch
104338-02   Security  Recommended       OpenWindows 3.5.1: libXt patch
105251-01   Security  Recommended       OpenWindows 3.5.1: libXt Binary Compatibility Patch
**Retrieving Patches**
Patch-ID    Checksum status   Description
--------    ---------------   -----------------------------------------------------------------------
103558-10   checksum match    SunOS 5.5.1: admintool/launcher fixes plus various swmtool
103582-15   checksum match    SunOS 5.5.1: /kernel/drv/tcp and /usr/bin/netstat patch
103594-10   checksum match    SunOS 5.5.1: /usr/lib/sendmail fixes
103600-18   checksum match    SunOS 5.5.1: nfs, tlimod and rpcmod patch
103612-33   checksum match    SunOS 5.5.1: libc, libnsl, libucb, nis_cachemgr and rpc.ni
103630-09   checksum match    SunOS 5.5.1: ip ifconfig arp udp icmp patch
103663-08   checksum match    SunOS 5.5.1: libresolv, in.named, named-xfer, nslookup & n...
103680-01   checksum match    SunOS 5.5.1: nscd/nscd_nischeck rebuild for BIND 4.9.3
103686-02   checksum match    SunOS 5.5.1: rpc.nisd_resolv patch
[ ... 22 lines deleted ... ]
103901-08   checksum match    OpenWindows 3.5.1: Xview Patch
104338-02   checksum match    OpenWindows 3.5.1: libXt patch
105251-01   checksum match    OpenWindows 3.5.1: libXt Binary Compatibility Patch
            ** Installing all patches without checking them first **
            ** can have negative consequences. I am assuming that **
            ** you know this, and think that all of these patches **
            ** are a good idea. Using the -F option will turn off **
            ** this message.                                       **
Which patches do you want to install (all/none/list of patches) all
**Installing Patches (this can take a while)**
Patch-ID    Install status    Description
--------    --------------    -----------------------------------------------------------------------
103558-10   Patch installed   SunOS 5.5.1: admintool/launcher fixes plus various swmtool
103582-15   Patch installed   SunOS 5.5.1: /kernel/drv/tcp and /usr/bin/netstat patch
103594-10   Patch installed   SunOS 5.5.1: /usr/lib/sendmail fixes
103600-18   Patch installed   SunOS 5.5.1: nfs, tlimod and rpcmod patch
103612-33   Patch installed   SunOS 5.5.1: libc, libnsl, libucb, nis_cachemgr and rpc.ni
103630-09   Patch installed   SunOS 5.5.1: ip ifconfig arp udp icmp patch
103663-08   Patch installed   SunOS 5.5.1: libresolv, in.named, named-xfer, nslookup & n
103680-01   Patch installed   SunOS 5.5.1: nscd/nscd_nischeck rebuild for BIND 4.9.3
103686-02   Patch installed   SunOS 5.5.1: rpc.nisd_resolv patch
103690-05   Patch installed   SunOS 5.5.1: cron/crontab/at/atq/atrm patch
[ ... 21 lines deleted ... ]
103901-08   Patch installed   OpenWindows 3.5.1: Xview Patch
104338-02   Patch installed   OpenWindows 3.5.1: libXt patch
105251-01   Patch installed   OpenWindows 3.5.1: libXt Binary Compatibility Patch
```

A Brave New World: UNIX Developers in an NT Land

by anonymous

Editor's note: The author believes this article could jeopardize his position with his employer. Therefore, I am allowing it to be published anonymously. Communicate with him via the email address below.

<sage_art@hotmail.com>

Take a bunch of developers used to the UNIX programming environment, move some of them over to NT, and you end up with less productive programmers who grumble a lot. Here are some of the problems we faced and what we were able to do about them and, in some cases, what we weren't able to do about them. No claim is made that our solutions or workarounds are universally the best ones or even the best one for any developers you might be supporting who are making the same crossing into a new world.

Shells and Stuff

MS-DOS's `COMMAND.COM` is reviled by many, and `CMD.EXE` (the default NT command shell) isn't a whole lot improved. This is too bad because there have been many shareware and freeware replacement command shells for `COMMAND.COM` that have been available for years (my favorite from a past life was the `4DOS/4NT/4OS2` shells from JP Software), and it would have been nice if `CMD.EXE` would have learned yet more from them. That's not to say there aren't a few improvements: for example, `CMD.EXE` windows support using the mouse to initiate marking text for cut and paste (although you have to use `ENTER` to end the marking, and you can mark only rectangles), and `TAB` does file completion, provided you tweak the right magic registry settings.

(Sure, uh huh, NT is much easier to administer than UNIX; after all UNIX has those horrible, obscure text configuration files, but NT has nice GUIs and, oh, by the way, equally obscure registry settings. Moral: obscurity isn't inherently a UNIX vs. NT thing.)

Some of us use `CMD.EXE` anyway, but others use a freely distributable port of `tcsh`. Although there are supported commercial packages like the MKS toolkit that include various shells, UNIX utilities, etc., so far we've opted to go with freely distributable ports of various utilities. Those of you who remember ports of various UNIX tools to MS-DOS (with considerable code rewriting necessary in the port effort and often with much functionality lost) shouldn't let those memories unfairly prejudice you against the ports that are now available. Porting to Win32 at the very least allows not having to mangle 32-bit code back into 16-bit code, and although the NT POSIX subsystem is incomplete, it's adequate for some utilities, and Win32 console mode is adequate for others. Even a fairly good freely distributable port of emacs19 is available, as are good ports of vi and derivatives.

Having a nice directory full of such gathered stuff on a server and exporting it to everybody reduces some of the grumbling. And that brings us to the next topic.

File Sharing

Fundamentally, we saw two choices between what protocols were going to flow over the network: NFS, which we all know and just love, or SMB, which stands for Server Message Block and is the foundation of Microsoft LAN Manager and subsequent native Windows and NT file sharing for both peer/peer and client/server. If we went primarily with NFS on the network, we would have to buy NFS clients for all NT systems and possibly NT NFS servers for at least some of the NT systems. The pro was that then

everybody could talk to everybody; both UNIX and NT systems could act as servers and as clients as long as we were willing to spend the money. The cons were the money and that we had heard claims of instability with adding NFS to NT, claims we weren't interested in confirming or denying ourselves.

We decided to go with SMB. Both commercial and freely distributable SMB servers exist for UNIX; we opted to try out Samba, which falls into the latter category, to simplify administration–anybody who wanted to try it out on a UNIX system could; we didn't have to count heads to see how many licenses we needed to buy. Samba enables the UNIX system to act as a fileserver (and print server, too, although we are not using that functionality); from an NT system, accessing a share (the term "share" is roughly equivalent to an exported filesystem specified in `/etc/exports` for NFS) on Samba server is just like accessing a share on an NT system.

The con with going with Samba is that there's no similarly transparent way for an NT system to serve files to a UNIX system. Samba supplies `smbclient`, whose UI is much like the standard UNIX FTP client. So on a UNIX system you can get and put files to an NT fileserver, but this is a bit awkward.

In any case, in practice, this inability to access NT shares transparently from a UNIX system has not been a major problem for our development workflow. For rarely accessed files, people will use smbclient; for more commonly accessed files, we can have the NT system serve the files via HTTP. One major reason we haven't stressed too much about transferring files back and forth is that our revision control system handles that without needing to have shared network filesystems.

Source Code Revision Control

Source code revision control systems are nearly as touchy an issue as favorite programming editors. They are actually worse in that you need to impose the same system on entire groups of developers, whereas individuals within that group can use their own favorite editors in semipeaceful coexistence with only occasional firefights. We had an in-house system that evolved over years with which developers weren't entirely happy, but at least they were used to it. But adapting it to work with NT seemed to be an intractable problem not worth butting our heads up against. We solicited opinions, looked for Web pages, read vendor glossies, even had some small, hungry companies come give us presentations. Finally, we opted to adopt Perforce. Their Web site will provide all the glowing points you want, but we particularly liked the number of supported platforms for both client and server, efficient use of network bandwidth across WANs (single TCP connection per client invocation), atomic changes, ease of writing scripts to add functionality, and the lack of requiring that you structure your development cycles and workflow the way they think you should. So far, we have no major regrets after months of usage.

A little gotcha about NT that we noticed only in the context of the source code revision control system is the annoying case insensitive but case-preserving semantics of NT filesystems. If developer #1 checks in `\lib\foo.c` from an NT system and developer #2 checks in `\LIB\bar.c` from another NT system, developer #3 on an NT system, who gets top of trunk, will see both files in the same directory, with the exact case of that directory's name being either lib, LIB, or possibly something else if developer #3 already had the directory created. But a developer on a UNIX system ends up with two files in two different directories and is left to wonder why he can't perform a full build any more. Perforce suggests several workarounds for this problem, one of which works fine for us, but it's nevertheless an annoyance.

A little gotcha about NT that we noticed only in the context of the source code revision control system is the annoying case insensitive but case-preserving semantics of NT filesystems.

With a little bit of work in setting up the software environment, NT can become a reasonably tolerable environment for software development . . .

Working Remotely

We have a fair amount of telecommuting. Most people work only occasionally from their homes, but several people live hundreds of miles away and are only in the office a few times a year. Being able to access resources across a dial-up session is therefore important. UNIX provides flexibility in how to access resources remotely: you can run everything on your local home system and go across the dial-up for remote filesystem access, or you can remotely login to a system at work (rlogin/telnet) and run text mode applications and tools or even X applications and tools, if you have the patience. NT supports the former just fine (SMB on TCP/IP on PPP is no problem), but doesn't support the latter out of the box. We knew there existed various commercial packages that support remote access for Windows by duplicating the entire desktop that appears on the graphics monitor across the dial-up session to be displayed on the home system and that over the years their speeds have even come to be about as tolerable as, perhaps even better than, running an X app across a similarly low bandwidth network connection.

We never got around to checking if such packages work on NT, though, because somebody stumbled across commercial software from Ataman Software that served our needs nicely–a Telnet daemon that runs on NT and allows you to Telnet into it. This is not to be mistaken for the beta Telnet daemon that is available with one of the NT resource toolkits that Microsoft sells in book+CD-ROM form, which is horribly unstable. The Ataman Telnet daemon is pretty stable, and establishes a session running your favorite shell, within which you can run console mode programs. You can't run IDEs or GUI debuggers, but at least you can fire off compiles using a command line compiler front end. Even when at the office, the Telnet daemon is quite handy; you don't have to switch keyboards and mice and possibly even have to turn around and move your chair to use another NT system. Because NT supplies a Telnet client, remote login in all combinations works fine.

Conclusion

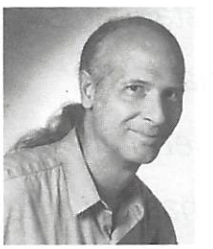
Developing on NT isn't nearly as bad as developing for DOS or Windows 3.x is. No horrible hacks like using the keyboard controller to forcibly reset the CPU between protected and "real" mode, no near vs. far pointers and plethora of "memory" models, no A20 gate, expanded, extended, and "high" memory to worry about in your applications; that stuff is gone. Networking and file sharing don't have to be bolted on after the fact any more. With a little bit of work in setting up the software environment, NT can become a reasonably tolerable environment for software development, even for UNIX-biased developers – a backhanded compliment, to be sure, but a compliment nevertheless.

[*Author's note*: A search engine like AltaVista should suffice to help you find vendor Web sites, distribution sites for freely distributable software, etc.]

[*Disclaimer*: Mentions of products should not be taken as endorsements. The author has no commercial relationships with vendors of add-on products mentioned other than as a paying customer in some cases.]

musings

Recently, I labored at removing barbed wire fences. These rusty relics once separated one ranch from another and stopped cattle from disappearing into the wilderness. It cannot be said that the fences prevented the cattle from destroying the land they grazed, which will take hundreds of years to recover. Before there were cattle, the land was covered in knee-deep grass, and even the trees were more abundant and varied. Now the land is rocky and barren.



by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V.*

<rik@spirit.com>

The fences in my backyard no longer correspond to any boundary. I am removing them so that my neighbors don't get confused and decide to build a real fence along the old, partially fallen down, fence line. I like the illusion fenceless backyards bring–that your house simply sits on the land, animals can move uninhibited from one yard to another, and there are no visible boundaries.

There are times and places for boundaries. My house serves as a boundary against bugs, critters, and weather, as well as surrounding and protecting my possessions. I have the opportunity to control who enters my house and how long they may stay. More than my yard, the house defines my true boundary–the yard is a buffer zone.

But there are other boundaries. The history of computers is replete with boundaries as artificial as any other made by human beings. And these boundaries have not served us well. Nor, in the long run, do they serve the vendors who created them.

Fences

Early computers were standalone by nature. It was quite enough to build, design, and operate a single mainframe, and nobody expected that one manufacturer's product could exchange data with another. In some cases, it was hard to exchange data even between the same makes of computers–unless you consider punched cards or paper tape efficient. Over time, standards for magnetic tape evolved, making it possible to exchange data between computers.

Rather than making it easy to connect computers, vendors strove to make them different. Repeat business is the foundation for any enterprise, and having customers who could buy their software, supplies, and replacements from only one vendor guaranteed repeat business. IBM and DEC had their own *terminals*. Legal battles were fought against vendors who dared to build compatible components such as memory or disks. Margins, the markup percentage, stayed high.

Revolution

Microcomputers created an avalanche of change. The earliest microcomputers were practically useless–interesting, but not good for getting any work done. There was no software.

One of the earlier successes was an operating system called CP/M, originally written on DEC hardware and ported to the Intel microprocessor. If you had a floppy drive that was aligned correctly, you could load CP/M, use software, and exchange data with other people and their computers. The key here was a combination of hardware (the floppy drive and the processor) and the operating system, CP/M.

IBM changed everything. I was working with one of the early microprocessor vendors when the PC came out. The engineers and I laughed when we learned that PC power supplies were blowing up. The VP of marketing overheard us and sternly reminded us that IBM could afford to give away PCs, and they would one day own the market.

Over time, the standards process, especially the excellent Internet standards process, has become more bogged down with vendor battles. Vendors plot to create a "standard" that will directly benefit their own designs or work to the detriment of a competitor.

Well, not quite. The PC created a de facto hardware standard. Once the BIOS had been successfully reverse engineered, PC compatibles could be built, and competitors almost shut out IBM from the market it had created. Margins today on desktop PCs are razor thin. For consumers, this is a great deal.

The basis for software compatibility was MS/DOS, a simple operating system that was chosen because the owners of CP/M had trouble with the IBM contract negotiations. Bill Gates didn't have an operating system, but he quickly bought the rights to MS/DOS, and changed the course of history. MS/DOS permitted software to be written for the new machines, making the new PCs actually useful.

Networks

As computers became more common, vendors conceived of wiring the computers together, creating a network. These early plans did not involve multiple vendors. The ARPAnet did manage to connect many computers, using one vendor's computer (BBN) as a front end. But vendors other than BBN were really not much involved in this process.

The early Internet had a much different model for breaking down boundaries. People would design software, use it, and propose that it was good enough for use by others. If the proposal was accepted, it was considered a standard. Although a document, the RFC, represents a standard, the operational basis is being able to communicate successfully with other systems using existing versions of the software.

Something similar happened with networking hardware. Although there are written standards, the real measure of compatibility for a network interface is being able to interoperate with other existing products. Of course, establishing this baseline was not as easy as I am perhaps making it sound. Again, standards evolved from working examples.

Having vendor-neutral standards has been very good for us. Standards foster competition, keep prices lower, and actually help us get work done. The focus is not on getting our computers and software to communicate, but on getting real work done.

Boundaries

Over time, the standards process, especially the excellent Internet standards process, has become more bogged down with vendor battles. Vendors plot to create a "standard" that will directly benefit their own designs or work to the detriment of a competitor. Not long ago I heard a salesperson say "they had just IETF'd" a competitor. He meant they had passed through a committee a "standard" that would directly benefit his company and hurt a smaller competitor.

In this case, standards become boundaries that help only certain vendors.

Another way of creating boundaries is to create a de facto standard, but not publish it. Microsoft has become a master of this technique by simply not publishing interfaces. The NT API is not published–the Win32 subsystem API is layered on top of the unpublished NT API. SMS (Simple Management System) requires the use of Microsoft's SQL server–and they refuse to publish schemas that would make other databases useful.

UNIX vendors have played this game, but in a different way. Workstations and servers were designed so that the only source for add-on memory or disks was the vendor. That has changed to a large degree, and there are second sources for most UNIX workstation, and even server, add-ons.

Even worse, UNIX vendors fought to create and maintain many minor differences in the versions of UNIX used. Although this may have appeared a religious war, it was not, no more than the Bosnian conflict is really about Christians and Moslems, but more about a surfeit of people and a scarcity of resources. Religion provides a useful demarcation, and the losers will no longer compete.

At this time, I would be happy to have network and administrative interfaces that work reliably between all versions of UNIX. The networking does work well, really, but the administration is another matter. NT strives to have coherent administration, but only among Microsoft products. Microsoft strives to create artificial barriers between other vendors' products and its own.

I don't really have any solution to offer. Tearing down old fences made me think about the other artificial boundaries that have been created in the world of computers, software, and operating systems. Achieving interoperability is not easy, but history has shown us it can be done. When it is done, business flourishes (PCs, the Internet), and when it is not done, businesses often suffer (IBM and DEC).

Achieving interoperability is not easy, but history has shown us it can be done.

In the world of security, it is a truism that if customers do not demand security from vendors, they will not get good security. Perhaps we should also be asking for real standards and products without artificial boundaries.

Will this be good for vendors, too? Economists tell us that a nation's economic success can be measured by its product. I disagree. A nation's wealth is measured by exchange–the exchange of products and services. If no one buys your product, you will have a valueless product. History has shown us that products that embrace standards have done better, that is, contributed more to exchange, over time than closed, proprietary products. Standards-based products are easier to exchange.

Over time, I believe that vendors who create artificial boundaries will suffer, as they have in the past. I just don't like dealing with them while this happens. Do your part. Insist on open standards. And I'll keep on tearing down old fences.

a conversation with Russ Cox

Russ Cox has been a world-champion-level computer programmer for several years. He has attended several USA Computing Olympiad training camps and a world championship tournament in Holland. He also has been one of the luckiest secondary students in the world: his part-time job is at AT&T Bell Labs. This conversation between Russ and Rob Kolstad was held via email during September of 1997.

Rob You've been associated with USENIX for several years now via the USA Computing Olympiad (USACO). Have you spent lots of time on USACO? Have those experiences had any value?

Russ Yes to both questions. Over the past three years, I've spent five weeks at national or international programming contests as well as time grading lesser ones. I attended the USACO training camp and final round, held at the University of Wisconsin, Parkside, in 1995, 1996, and 1997. In 1995, I had the opportunity to represent the United States at the International Olympiad in Informatics (IOI), held in the Netherlands. Shortly after Thanksgiving, I will be travelling with the US team to South Africa for the 1997 IOI.

In addition to time spent at contests and preparing for them, I graded our two Internet contests this year as well as the USACO Competition Round. Grading is somewhat automated, but still requires 15 to 20 hours of human time for each contest. I hope to fully automate the process for the coming year.

These experiences have had tremendous value for me. I had a great time at the USACO training camps during the past three years. At both the USACO training camp and the IOI, I've had a chance to meet and get to know bright computer programmers from all over the United States and around the world.

At the same time, I've learned quite a bit about algorithms and programming style from the contests. All the formal instruction I've ever had about topics like algorithmic complexity or geometric algorithms has come from USACO's training camp. Furthermore, discussions from training camp, whether during a lecture or over lunch, have sparked self-study of myriad topics from hashing to finite state machines to machine instruction timings.

Finally, there was 1996. As I mentioned before, I made the IOI team in 1995. Thanks to a combination of other distractions and overconfidence, I didn't prepare myself enough for the 1996 final round and performed extremely poorly. As a result, I didn't make the IOI team that year. I worked hard in preparing for 1997's training camp and happily made the IOI team once again. The whole sequence was a great lesson in both humility and persistence.

Rob And now you're headed to college?

Russ Yes. I just graduated from high school last June, and am starting my first year at Harvard. I wanted a liberal arts school so that I'd have a wide range of nontechnical courses to choose from. If the technical sides are ever lacking, it's relatively painless to cross-register for classes at MIT.

Rob How will you choose a major?

Russ I'm not really sure how I'm going to choose a major. I'm leaning towards computer science right now–and math is definitely in the running–but anything is possible. No matter what the major, I plan to take all my electives outside it. Two years ago I was fully convinced that courses like History and English were just not for me and didn't pay much attention to them. Some great teachers during the last two school years turned me around completely, and I definitely expect to take a broad variety of courses.

Rob What do you anticipate will be the high points at Harvard? Low points?

Russ My high school courses were great, but often left me pursuing further exploration of the material on my own. The library was often useless; if I knew a teacher who might have information on the subject, that often helped. But usually I never

found what I wanted to know. I'm looking forward to a much richer font of information at Harvard.

Furthermore, I'm excited about the chance to meet and interact with so many bright people from so many different places. USACO, IOI, high school, and my various summer jobs have shown me that I enjoy being around really smart people from all over. One of the highlights of the past school year was arriving at school an hour early each day and sitting around with a handful of friends having philosophical discussions about current events, ancient history, or even a random Simpsons episode that someone had brought in. When I visited Harvard, I got the impression that this was a very realistic expectation for the future as well.

At the same time, I'm hoping that the environment shift won't be too much of a shock. I haven't been thrown into a wholly new environment in six years, but rather have been task switching between four or five established ones. I'm leaving all of these behind to explore a new one. I don't expect it to be a large problem, but it's a skill I'll have to relearn. The same applies on a smaller scale to my computing environment, but again, I expect it to be more a chore than an insurmountable obstacle.

Rob Have you been able to apply any of your computer skills in school or in the workplace?

Russ Yes. I spent a good part of my senior year in high school setting up an Internet gateway and an email subsystem for the school and even got course credit for it. In addition, I've spent the last three summers working for Bell Labs and now AT&T Labs–Research.

Rob What did the Internet gateway for school entail? Did you learn much?

Russ I started with a stock Linux system. Initially, I worked quite a bit on replacing some of the more complex Linux daemons with simple, more secure, home-grown ones. The whole login process was shell scripts augmented by a couple five-line C programs, complete with md5sum-based challenge/response authentication.

I also hacked up 70% of a sendmail replacement inspired by Dave Presotto's `Upas` mailer, but ultimately concluded it was too big a job. I had everything done save walking the outgoing mail queues (we were still passing outgoing mail to BSD sendmail), but found that good error recovery was increasingly a problem. If I had been there for another year, I would probably have stuck it out with my mailer and login programs, but I decided the system had a much better chance of surviving without me if it was running daemons that were at least well understood. Unfortunately, I didn't really have enough time to produce a robust system. Now the only extensive shell scripts that remain in use are the ones that perform local delivery to Novell Netware.

I learned an incredible amount about TCP/IP, mail routing, and shell scripts, as well as way too much about the Windows 95 TCP/IP stack. I still get a phone call once in a while when things break and the student who took over for me isn't available.

Rob And at work? You mentioned AT&T Research and Bell Labs.

Russ For the last year or so, I've worked with cable modem trials for AT&T Labs–Research. We telecommute using cable RF downstream and 28.8kbps POTS modems back upstream. The speed is really nice. If you play with the TCP window sizes, you can get a full two megabits per second datastream into a PC. The result is incredibly nice Web/Ftp access–you begin to notice sites that are "only" connected to the net via ISDN or a T1. You get hit whenever you want to upload something, but it's easy enough to avoid.

Russ Cox

"I've internalized the UNIX "do one thing well" philosophy to a much greater extent than I had three years ago. This has made my programs ultimately simpler, more concise, and yet much more useful."

Besides the cable modem work, I do a lot of system administrivia. In addition, I've been involved with maintenance of the servers that handle challenge/response authentication for our Internet firewall. There are also lots of small "one afternoon" projects: augmenting Research UNIX's `lp` to be able to send jobs to BSD `lpr` hosts, writing a quick and dirty SNMP client to maintain modem racks remotely, things like that.

Finally, I multitasked work for AT&T this summer against playing with Brazil, Bell Labs's internal successor to Plan 9. I have it running on a laptop and recently have been playing with DHCP and other issues related to portable computing. Mostly, it's involved hacking at the IP stack and other network-related programs; it's a lot of fun.

Rob How have these experiences been of value to you?

Russ First of all, I've learned incredibly much about dealing with users. At school, I supported the entire faculty and student body, and at AT&T I've been involved with technical support for others involved in the cable modem trials. The skills you learn in user support–clearly defining the problem, unambiguous communication, asking effective questions, patience–are also helpful people skills for life in general.

The second benefit has been to my programming. Since I first installed the public release of Plan 9 in April of last year, I've read quite a bit of the kernel and other sources. Reading quality code has improved my programming skills amazingly. There is just no comparison between programs I wrote two or three years ago and the programs I've written recently. Furthermore, thanks to both reading code and, more importantly, interacting with so many people from the pre-split Bell Labs computer science center, I've internalized the UNIX "do one thing well" philosophy to a much greater extent than I had three years ago. This has made my programs ultimately simpler, more concise, and yet much more useful.

Rob Do you participate in any activities outside the computer world?

Russ Yes. I'm currently an assistant scoutmaster with the Boy Scout troop in town. As a scout in that troop, I earned the Eagle Scout rank and spent my last five years in various leadership positions. In high school, I was the editor of the school newspaper and managed the baseball team. I also sang in the school choir. For the last five years, I've worked for some fraction of each summer as a lifeguard at a Boy Scout camp in upstate New York. I put in a full season this year–five weeks– as a good way to end my time there, at least for now.

Rob Would you like to share any dreams you have with our readers?

Russ Ultimately, I don't know what I want to do when I get out of school. It might be research or teaching or something entirely different. I just don't know. There are some persistent dreams, however:

- A year or so ago, I had this one moment of great clarity where I was convinced that P == NP, but I don't remember the reasoning behind it.
- A friend and I have been tossing around the idea of writing a book about the kind of programming issues that are involved in programming contests and that genre of problems in general.
- Someday, I'm going to bowl a 300. A bunch of high school friends and I went bowling semiregularly over the summer, and I bowled a 180 my last time out.
- Somewhere in the future, I'd like to grab a couple friends and hike the entire Appalachian Trail. It runs from Maine to Georgia, and people doing the entire thing usually take several months.

using C++ as a better C

In this column, we'll spend some time talking about C++ classes, structs, and unions, illustrating several points unrelated to object-oriented programming.

by Glen McCluskey

Glen McCluskey is a consultant with 15 years of experience and has focused on programming languages since 1988. He specializes in Java and C++ performance, testing, and technical documentation areas.

<glenm@glenmccl.com>

Type Names

In C, a common style of usage is to say:

```
struct A {
   int x;
};
typedef struct A A;
```

after which A can be used as a type name to declare objects:

```
void f()
{
   A a;
}
```

In C++, classes, structs, unions, and enum names are automatically type names, so you can say:

```
struct A {
   int x;
};
void f()
{
   A a;
}
```

or:

```
enum E {ee};
void f()
{
   E e;
}
```

By using the `typedef` trick, you can follow a style of programming in C somewhat like that used in C++.

But there is a quirk or two when using C++. Consider usage like:

```
struct A {
   int x;
};
int A;
void f()
{
   A a;
}
```

This is illegal because the `int` declaration `A` hides the `struct` declaration. The `struct A` can still be used, however, by specifying it via an "elaborated type specifier":

```
struct A
```

The same applies to other type names:

```
class A a;
union U u;
enum E e;
```

Taking advantage of this feature, that is, giving a class type and a variable or function the same name, isn't very good usage. It's supported for compatibility reasons with old C code; C puts structure tags (names) into a separate namespace, but C++ does not. Terms like "struct compatibility hack" and "1.5 namespace rule" are sometimes used to describe this feature.

Bit Field Types

Here's a small difference between C and C++. In ANSI C, bit fields must be of type "int," "signed int," or "unsigned int." In C++, they may be of any integral type, for example:

```
enum E {e1, e2, e3};

class A {
public:
   int x : 5;
   unsigned char y : 8;
   E z : 5;
};
```

This extension was added in order to allow bit field values to be passed to functions expecting a particular type, for example:

```
void f(E e)
{
}

void g()
{
   A a;
   a.z = e3;
   f(a.z);
}
```

Note that even with this relaxation of C rules, bit fields can be problematic to use. There are no pointers or references to bit fields in C++, and the layout and size of fields is tricky and not necessarily portable.

Anonymous Unions

Here's a simple one. In C++, this usage is legal:

```
struct A {
   union {
      int x;
      double y;
      char* z;
   };
};
```

whereas in C, you'd have to say:

```
struct A {
   union {
      int x;
      double y;
      char* z;
   } u;
  };
```

giving the union a name. With the C++ approach, you can treat the union members as though they were members of the enclosing struct.

Of course, the members still belong to the union, meaning that they share memory space and only one is active at a given time.

Empty Classes and Structs

In C, an empty struct like:

```
struct A {};
```

is invalid, whereas in C++, usage like:

```
struct A {};
```

or:

```
class B {};
```

is perfectly legal. This type of construct is useful when developing a skeleton or placeholder for a class.

An empty class has size greater than zero. Two class objects of empty classes will have distinct addresses, as in:

```
class A {};
void f()
{
   A* p1 = new A;
   A* p2 = new A;
   // p1 != p2 at this point ...
}
```

There are still one or two C++ compilers that generate C code as their "assembly" language. To handle an empty class, they will generate a dummy member, so, for example:

```
class A {};
```

becomes:

```
struct A {
   char __dummy;
};
```

in the C output.

Name Hiding

Consider this small example:

```
#include <stdio.h>
int xxx[10];
int main()
{
   struct xxx {
      int a;
   };
   printf("%d\n", sizeof(xxx));
   return 0;
}
```

When compiled as C code, it will typically print a value like 20 or 40, whereas when treated as C++, the output value will likely be 2 or 4. Why is this? In C++, the introduction of the local struct declaration hides the global `xxx`, and the program is simply taking the size of a struct which has a single integer member in it. In C, `sizeof(xxx)` refers to the global array, and a tag like `xxx` doesn't automatically refer to a struct. If we said `sizeof(struct xxx)`, then we would be able to refer to the local struct declaration.

Next month: memory allocation.

debugging in Java

by Philippe Kaplan

Philippe has worked for Dyade, a French applied research center, since he received his PhD in 1992. He specializes in software tools, and is now working on a graphic toolkit written in pure Java.

<Philippe.Kaplan@sophia.inria.fr>

A new language like Java means new features, new programming techniques, and . . . a brand new generation of bugs. Articles about Java are now common and help to take advantage of the powerful features of the language. However, I have seen few articles that help debugging in this language, as if the debug techniques remain the same across the different programming languages.

Believe me, this is wrong. The usual debug techniques are still valid (actually, it is hard to break developer habits in this area) but new techniques are available to tackle new kind of problems. We can improve our efficiency in debugging by taking advantage of Java features.

Debugging with Objects

Non-object-oriented programs are structured sets of data and instructions. These are quite straightforward to debug if you know roughly where the bug is located. Just set breakpoints in the guilty area and run within the debugger. When the program hits the breakpoint, it stops, and the context can be studied to understand and discover the bug.

Things are different in object-oriented languages like Java. Breakpoints are still set on instructions; however, instructions belong to object instances. So breakpoints stop within all instances, and even if we know within which instance the bug is located, there is no way to track down only this instance. For example, assume we create an instance for each dynamic point of a moving graph and there is a bug in some of the points leading to an unexpected peak in the graph display. We know which ones among hundreds of points are faulty, but not how to trace only them and then detect when they go out of bounds.

Threads make things worse. The developer is misled by the source code, which appears basically single-threaded. The developer sees one line of the program, but threads are actually running simultaneously at different points of the same program. The puzzle is real: it is difficult to visualize several threads running at different places and on different instances of the same class. Breakpoints are hit, but by which instance, within which thread?

Let's forget about high-level debuggers, because today they are still far from being powerful enough to tackle our problem. How can we debug a threaded object with only println statements in the source code ? Putting `println` in the class definition comes to mind, but is not enough. It will trace the method called by all instances, and we want to trace only a particular instance.

Java documentation reveals a heavier hammer, namely, `Runtime.traceMethodCall()`, which prints a line for each method call of all objects in all threads. The next section discusses this tracing method. However, this is still maladapted for our purposes.

The solution appears in Java 1.1, with the inner classes. When the faulty instance is created, we override on the fly the method to be traced so it prints a message. For example, given the class Test where the method monitor has to be traced only on the instance `obj1`:

```
public class Test {
public Test(int x) {
  ...
}
```

```
  public synchronized void monitor(String label) {
    ...
  }
}
// (somewhere else, in the program)
Test obj1 = new Test(1); // I'm sure there is a bug in this object
Test obj2 = new Test(2);
Test obj3 = new Test(3);
```

We subclass on the fly the suspected instance and override the method to be traced. The `obj1` creation becomes:

```
Test obj1 = new Test(1) {
  public synchronized void monitor(String label) {
    // print a trace
    System.err.println("debug: <" + Thread.currentThread().getName()
            + "> is calling " + this + ".monitor(" + label + ")");
    // call the real method
    super.monitor(label);
  }
};
```

During execution, each time the method is called on the specified object, a message line with the calling thread name, the object, the method name, and its argument are printed out synchronously, e.g.:

```
debug: <main> is calling Test$1@80f0dfd.monitor(foo)
```

Only the suspect instance is traced, and this object still behaves exactly as it did (the trace side effect excepted). The synchronized modifier ensures that the lock is acquired before printing the trace, so no other thread may interleave between the trace printing and the call of the method. Therefore, the message is indeed printed at the right time by the right thread.

Without the synchronized modifier, no assumption can be made about the calling thread. Obviously, it doesn't matter when the bug doesn't deal with threads.

If the debugger is required, just put a breakpoint at the method of the inner class. The anonymous inner class (AIC) name is automatically generated by appending `$n` behind the original class name, where `n` is the number of the AIC in this class (e.g., `Test$1`). Here is an example of a session under the debugger:

```
$ jdb Test
Initializing jdb...
0x40786388:class(Test)
> stop in Test$1.monitor
Breakpoint set in Test$1.monitor
> run
run Test
running ...
main[1]
Breakpoint hit: Test$1.monitor (Test$1:6)
main[1]
```

The debugger is stopped before the call `super.monitor(label)`.

Remote Tracing

Java runs threads and is network aware. The Java debugger (jdb) is so well integrated it allows remote debugging of Java applications in a separate thread. This is quite a new concept for developers (like me) who come from C and its symbolic debugger. Here is an example of a new debugging trick.

The Java debugger (jdb) is so well integrated it allows remote debugging of Java applications in a separate thread. This is quite a new concept for developers (like me) who come from C and its symbolic debugger.

Sometimes, when everything else fails, we need the ultimate debugging tool: `traceMethodCalls(boolean)`. This function enables and disables tracing at runtime by toggling a debug switch of the byte code interpreter. Unfortunately, it traces all method calls (i.e., all objects in all threads). So the output is often too verbose to be really workable.

We are going to see how to let the user toggle the trace interactively without putting intrusive code in the application. Using this feature exactly when needed will reduce the volume of information produced.

The principle is quite simple. Because the trace switch affects all the environment, it can be toggled from a separate thread. Indeed, the debugger thread seems the best candidate because it works beside the application ones. So we run the Java debugger, and inside it we execute a standalone Java application that toggles the trace switch.

Compile the file Trace.java that contains:

```
public class Trace {
static boolean state = false;
public static void main (String a[]) {
  state = !state;
  Runtime.getRuntime().traceMethodCalls(state);
  System.out.println("method call trace "+ (state?"on.":"off."));
  }
}
```

This standalone application controls only the tracing device in the Java virtual machine.

Start the target program, with tracing and remote debugging enabled. `java_g` enables method tracing. The `-debug` switch opens an external access to the application that is dedicated to the remote debugger `jdb`.

```
$ java_g -debug myProg
Agent password=3hszsj
```

Start the remote debugger in another window. The password argument must be the one given by the application:

```
$ jdb -host localhost -password 3hszsj
```

Load the trace class (`jdb` window):

```
> load Trace
```

Proceed with the application. When you need trace, just type in `jdb`:

```
> run Trace
```

This activates the trace: all method calls are now printed out. To stop tracing, type in `jdb` again :

```
> run Trace
```

The trace is no longer activated, but the user is free to request the tracing again.

Note that the Trace class does not belong to the debugged application. The debugger loads this class by itself and executes it in its own thread on user request. The side effect of the command causes application threads to be traced. The target application does not need any change to be traceable. Could you imagine such a miracle with `gdb`?

Remote Thread Display

We debugged objects; we debugged method calls; now let's tackle threads. Threads are volatile. They appear, they disappear, they run, they wait inside or outside a monitor, they are suspended, and they may be numerous.

In a single debugger window, it is hard to track a thread life and even harder to follow the interactions between several threads. This is why I propose a simple package to display all threads and their status at regular intervals. (This package is going to be improved in order to provide more user control over thread execution and priority.) It works like a remote debugger, collecting information about threads on the target application.

To install the package threadDebug, download `threadDebug.zip` <http://www.inria.fr/koala/phk/java/threadDebug.zip>. (The source is available from <http://www.inria.fr/koala/phk/java/threadDebug-src.zip>). You don't need to unzip `threadDebug.zip`. Just set the `CLASSPATH` environment variable so that it also includes the zip file.

The package works like the remote tracing. The application is started with the `-debug` option, and the package uses the password given to establish a connection to the application, from another window. So here is a typical session. Start the program, with remote debugging enabled, e.g.,

```
$ java -debug myProg
Agent password=3hszsj
```

Note that the program may be an applet:

```
$ java -debug sun.applet.AppletViewer foo.html
Agent password=3hszsj
```

Start the remote thread displayer in another window:

```
$ java threadDebug.ThreadDisplay -host localhost -password 3hszsj \
  -refresh 2000
```

The last argument is optional and sets the automatic refresh time in milliseconds. The default refresh time is one second.

Several windows will pop up, one for each thread group that lists the threads of this group and another one for the groups. In addition, each change is printed sequentially on the screen. You should see something like Figure 1 on the facing page.

The refresh buttons start a manual refresh in windows, in addition to automatic ones. The tool displays active and inactive threads. Threads in "running" state are highlighted. As the application is going on, the "thread displays" reflect the states of all the threads involved in the application. This tool may help locate a bug by pointing out the threads that run the bug. It may also be useful to detect deadlocks and starvation.

Note also that on UNIX systems, the ctrl-backslash key (i.e., signal 3) prints a snapshot of all threads on stdout. The listing is more complete but less readable than the threadDisplay tool.

This tool, like the previous one, does not require any change in the target application to be enabled.

In a single debugger window, it is hard to track a thread life and even harder to follow the interactions between several threads. This is why I propose a simple package to display all threads and their status at regular intervals.

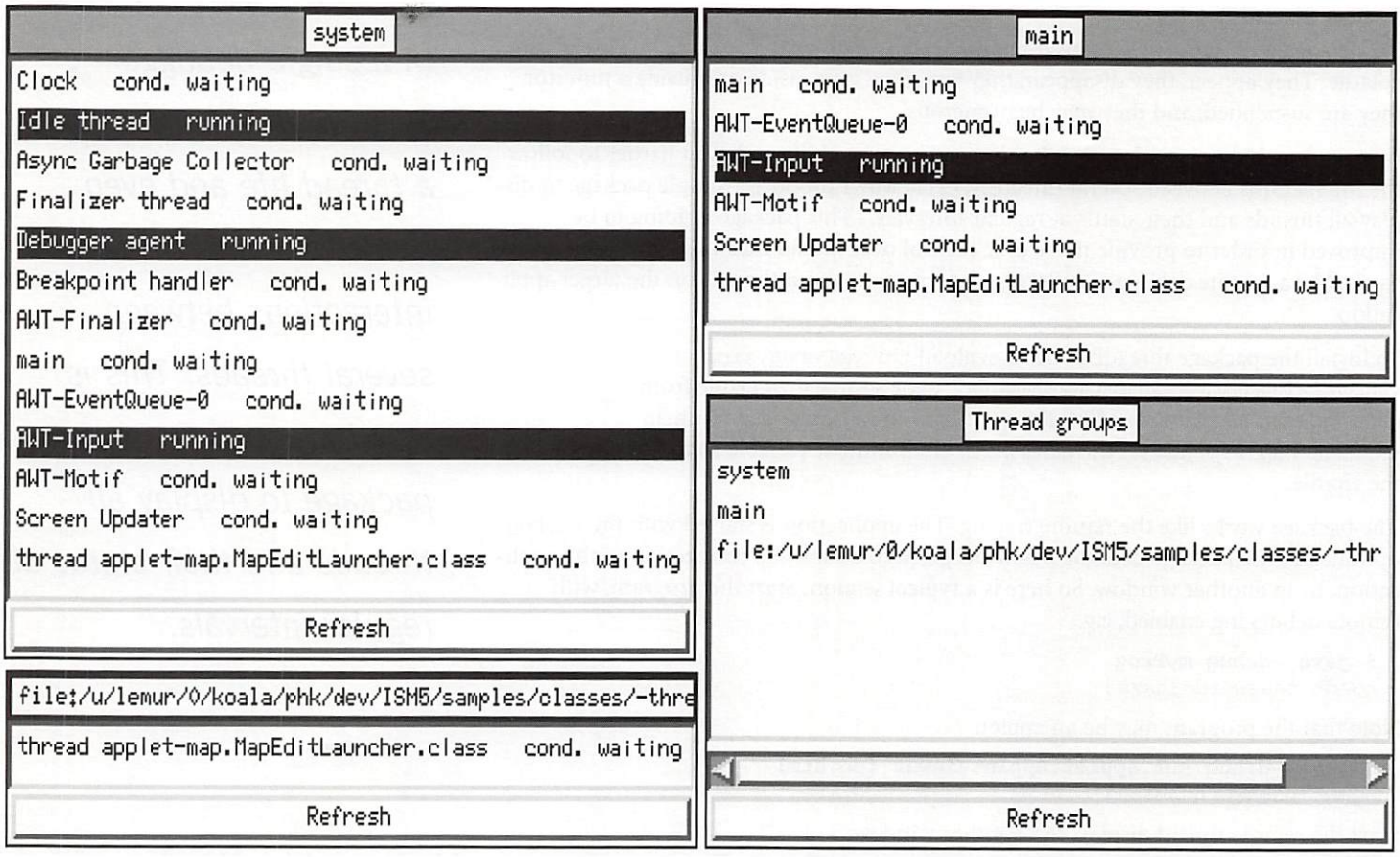


Figure 1: A thread debug display snapshot

Conclusion

Java, as a new multipurpose language, incorporates many hi-tech features. The object model together with the multithreading introduces a powerful new way of programming, but also a new class of bugs dealing with concurrent access to shared resources.

Fortunately, Java designers, aware of the importance of good debugging tools, have set specialized entry points into the language and the virtual machine to ease debugging. Actually, the official debugger, `jdb`, is only a user interface to the standard Java debug library.

With the ability to access debug functionalities from the language and the flexibility of an object-oriented model, we can create our own debug tools. These tools help to investigate particular situations, like we did here for objects and threads.

the webmaster: the virtual poet

Email from CGI Scripts

This time I want to show you the basics of creating an area on your site that sends email based on the information entered by a user. For my example, I'm going to show you the steps involved in creating the Virtual Poet.

by Dave Taylor

Dave Taylor is the president of interface design firm Intuitive Systems and author of many best-selling books on UNIX and Web page design. He's also CTO of The Internet Mall, the largest shopping site on the Net.

<taylor@usenix.org>

The concept is simple: the visitor to the site will enter his or her name, email address, and the name and email address of the recipient. Having specified that, the visitor will enter a poem, which will be mailed directly to the recipient. Of course, this is just a skeleton; otherwise you'll be asking why you wouldn't just enter the poem directly in an email message. But roll with me, eh?

Entering the Poem

Let's start with the entry screen–`index.html`– which looks like this:

```
<FORM ACTION=send-poem.cgi METHOD=post>
Your name:
<INPUT TYPE=text NAME=yourname SIZE=50>
<P>
Your email address:
<INPUT TYPE=text NAME=youremail SIZE=40>
<P>
Recipient name:
<INPUT TYPE=text NAME=theirname SIZE=50>
<P>
Recipient email address:
<INPUT TYPE=text NAME=theiremail SIZE=40>
<P>
Type in your poem here:
<TEXTAREA NAME=poem ROWS=7 COLS=60>
</TEXTAREA>
<P>
<INPUT TYPE=submit VALUE="send your poem">
</FORM>
```

It's a basic input form with five fields; the name and email address of the sender, the name and email address of the recipient, and the poem. If there's anything interesting to note about this, it's that I don't have a `NAME=` area on the final submit button, which means that the button value won't be sent to the receiving script (`send-poem.cgi`, as specified in the `ACTION` clause of the `FORM`).

Receiving the Poetry

So far, the CGI scripts we've written have received information via a `METHOD=get`, but that's not going to work with this particular scenario because we might receive more data than can be included on a URL line. In any case, it's more elegant.

To receive the information, we need to read it off the data stream. To do that, I'll utilize one of a set of useful C routines I've previously written and saved as `cgi-utils.c` (You can download a copy and read it yourself, if you're curious.)

There are some additional parts to the puzzle, of course, things that you need to get a full C program to work, but the gist of it has been presented here.

The basic strategy for using the CGI utilities library is to use the routine `get_cgi_environment()`, which returns the data stream exactly as transmitted. A typical value might look like:

```
yourname=Dave&youremail=taylor@intuitive.com&
theirname=Dave2&theiremail=taylor@netcom.com&
poem=roses+are+red+etc.
```

The information is packed for transmission. That's where the second routine comes in handy: `cleanup()` unpacks all the special encodings and lets you use the `valueof()` routine to extract specific values to the named variables. So you could easily have a simple line of C like:

```
printf("yourname = %s\n", valueof("yourname", cgienv));
```

Those three routines–`get_cgi_environment()`, `cleanup()`, and `valueof()`–are the heart of the `cgi-utils` library. Armed with them, we can get on to the actual program.

Reading the Environment

Here's the beginning of the actual `send-poem.c` program:

```
original_env = get_cgi_environment();
environment = cleanup(original_env);
```

Then we save all the variables into their own space for easier reference:

```
strcpy(yourname, valueof("yourname", environment));
strcpy(youremail, valueof("youremail", environment));
strcpy(theirname, valueof("theirname", environment));
strcpy(theiremail, valueof("theiremail", environment));
strcpy(poem, valueof("poem", environment));
```

That's the hard part. It's all an easy progression from here, much of which is codified in the routine `send_notification()`:

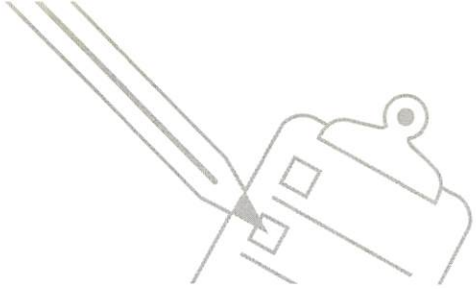
```
send_notification(yname, yemail, tname, temail, poem)
char *yname, *yemail, *tname, *temail, *poem;
{
  FILE *pd;
  char command[SLEN];

  sprintf(command, "/usr/sbin/sendmail -oi %s", temail);

  if ((pd = popen(command, "w")) == NULL) {
   printf("Couldn't open pipe '%s'\n", command);
   exit(0);
  }

  fprintf(pd, "Reply-To: %s (%s)\n", yemail, yname);
  fprintf(pd, "To: %s (%s)\n", temail, tname);
  fprintf(pd, "Subject: A Virtual Poem from %s\n", yname);
  fprintf(pd, "\n");

  fprintf(pd, "%s has a poem for you:\n\n", yname);
  fprintf(pd, "%s\n\n", poem);

  fprintf(pd, "\n\nThis poem brought to you by The Virtual
                                  Poet.\n");

  pclose(pd);
}
```

There are some additional parts to the puzzle, of course, things that you need to get a full C program to work, but the gist of it has been presented here. With all this plugged in, the Virtual Poet site is up and ready to try.

You can visit <http://www.intuitive.com/CGI/poems/> for yourself and see what it looks like.

Expanding the Site

What would really make this cool, of course, would be to have the poem actually saved on the server and a "key" mailed to the recipient, who would then go to a specific spot on the site, enter the key, and receive a Web page that offers the poem in an attractive format. A number of virtual florists and electronic greeting card sites that do just this. It wouldn't be too hard: either you would create the Web page and save it directly when the user enters their poem, or save an intermediate data set, then parse and create the page on the fly. A little help from `cron` to expire files as they get more than a certain age, and you'd be ready to compete with some very expensive, carefully designed sites.



standards reports

The following Reports are published in this column:

Nick Stoughton welcomes dialogue between this column and you, the readers. Please send any comments you might have to: <nick@usenix.org>

An Update on Standards Relevant to USENIX Members

by Nicholas M. Stoughton

USENIX Standards Liaison

<nick@usenix.org>

Quo Vadis POSIX?

In July, a senior vice president from Hewlett-Packard, Larry Dwyer, was invited to make a presentation to the Sponsor Executive Committee (SEC) of the Portable Applications Standards Committee (PASC) on why they should stop producing POSIX standards. Essentially his argument was "POSIX has been a good thing in the past, but our customers are idiots who don't understand that POSIX is not one standard but many. As a result, the term "POSIX conformant system" has now lost any real meaning, and it's too expensive for us to keep up."

Now it may not sound like too good a strategy to go to your customers and tell them that they are idiots. It may not sound like a good strategy to go to a bunch of dedicated standards producers and tell them they are wasting their time. But we are people not easily offended, and we actually listened to him. We are even acting on some of his comments. The SEC came up with eight items for further discussion:

1. Demonstrate a clear commitment to backward compatibility, whenever possible, for POSIX.1 and POSIX.2.
2. Change the structure of POSIX.1 and POSIX.2 and their amendments so that options in new amendments do not require vendors to revise their systems that do not claim to support those options.
3. Based on an architectural perspective, develop a new set of POSIX named "profiles." Develop a guideline, a touchstone, to determine what subjects are and are not "POSIX."
4. Define a limit to amendments to POSIX.1 and POSIX.2. Raise the hurdle to add new things.
5. Collaborate with The Open Group for marketing, branding and conformance testing, while retaining open participation.
6. Work with IEEE to provide for corporate representation in the IEEE ballot process, independent of individual member balloting rights.
7. Identify and fix bugs in process, e.g., stale balloting groups and how to freshen them in timely fashion.
8. Provide additional mechanisms for soliciting broader input on new Project Requests (e.g., 3 month delay for community input, etc).

Most of these points have now been discussed and actioned. In some cases, the discussion demonstrated that we rejected the idea; for example, there will be no limit imposed on amendments to POSIX.1 or POSIX.2. New features will generally be made optional, with minimal effort required from a vendor that does not want to support that option; all it has to do is to redefine the value of `POSIX_VERSION` and it conforms to the new version of the standard! In general, we do not expect vendors to ignore all new optional features, but the let out is there.

Working to try to improve the process is like trying to push gravel up a steep hill. You think you are making headway, till you realize that most of the stones have slipped back down the hill again!

Corporate representation, for example, is such an anathema to so many IEEE staff that many of us doubt it will ever happen. But we continue to lobby for it. In reality, very few of the "individuals" who are working on POSIX standards really are working independently; they are all on company time and pay. When they change jobs, they often stop coming, or stop responding to ballots. Individual membership should never be prevented, but it seems ridiculous that people who do not want to be involved cannot hand over responsibilities to someone who does. Actually, that is one area where we appear to be winning; individuals can now opt out of a ballot group for the first time. And new members can join as observers but without the ability to cast a binding vote. Of course, the ballot review committee is unlikely to ignore input from someone who is able and willing to express an opinion, even if that opinion is not binding.

Part of the Dwyer argument was that we in PASC are producing more and more specialized options that few people want. But in the same breath, he complains that customers are demanding these options. If customers want them, then they'll pay for them to be implemented.

The "Dwyer Approach" has been an interesting exercise, but by far the best method to make sure that the standards we produce are useful and implementable is to join the working group, join the ballot group, and steer the standard in the right direction, rather than attacking the integrity of the individuals currently involved.

Revising POSIX.1 Classic

The POSIX.1 standard was first published in 1986 as a trial use standard, then again in 1988 as a full use standard. Subsequently, it was revised and republished in 1990 as an international standard (ISO 9945-1:1990). All standards in IEEE and in ISO are subject to a five-year review, at which point they must be reaffirmed, revised or withdrawn. We reaffirmed POSIX.1 in 1995 with a minimum of fuss. However, the time is fast approaching for the second of these review points, and we know this time we do need to revise the standard. A revision will have a number of effects; the entire document, including all the published amendments, is open for review. The result will not be one standard and a bunch of amendments (as it is now), but a single unified standard to which further amendments can be made.

The amendments that are expected to be finished by 2000, and therefore included in the new revised standard, are POSIX.1a (system API extensions), 1b (realtime), 1c (threads), 1d (advanced realtime), 1g (protocol independent interfaces, sockets etc), 1h (services for reliable, available and serviceable systems), 1i (realtime corrigenda), 1m (checkpoint/restart), and 1n (corrigenda). Now the astute among you may have noticed that less than half of this list is as yet actually published! How can we revise something that is only a few months old? Well, the answer is that with good management, we won't actually touch those parts; they will be written ready to go into the new standard. But before the revision, they are standards that amend POSIX.1; after it, they will simply be part of POSIX.1.

The Project Authorization Request (PAR) for this revision is going through the approval process right now, and limits the scope of the revision to just the functionality in the original document and the amendments; in other words, we can't go pushing new stuff into this revision. We can try to make sure that things work coherently throughout the standard; for example, error handling and option handling differ from amendment to amendment currently, and we could straighten this out.

The other big issue with the revision is: who is going to do it? If nobody comes forward willing to volunteer the time and effort, then this revision may never happen. It has a very finite time span. It must be finished by the end of 2000, or the old standard must be reaffirmed. It is unlikely that we would be allowed to reaffirm again, since there are now too many amendments. No volunteers, and POSIX.1 could just be withdrawn. It is possible that The Open Group might volunteer to assist in this, since they recognize the value of the standard. But even if it does, we need real users of the standard involved as well as the vendors. Please let me know if you can help!

The Open Group Distributed System Management

Nick Stoughton <nick@usenix.org> *reports on the September 1997 meeting in Boston, MA.*

The IT DialTone, as I have reported previously, is the new Open Group initiative to cause the creation of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy to use as the telephone. Work is now under way on phase one of this project. Phase one is intended to be a consolidation of what exists, trying to bring it all together into the common direction toward which we are aiming. To this end, a taxonomy of management services has been produced, and work is under way to revise the X/Open Manageability Guidelines (a document presented for other working groups to understand where and how to include management interfaces in the work they are producing).

Manageability is one of the key elements of the IT DialTone. Within the IT DialTone environment, management extends beyond the traditional confines of managing IT platforms and networks. Successful management for the IT DialTone must provide a comprehensive management capability that encompasses the full end-to-end viewpoint that is experienced by the application programmers and users of the environment.

Besides managing the infrastructure itself, it is also necessary for management to include the applications themselves, allowing them to be incorporated as manageable elements within the overall environment.

The table below enumerates a set of management areas and describes the environments in which they are applicable (i.e., a Managing Environment or a Managed Environment). In some cases, additional functions will be required to complete the management process. These additional functions may include manual processing.

This taxonomy is intended to provide a checklist against which the list of managed services can be validated for completeness. There is not a one-to-one correspondence between the management areas identified in the taxonomy and the proposed management services. Several issues may be addressed by a single service, and some issues may be instantiated across several services.

Further details about the IT Dialtone can be found at <http://www.opengroup.org>.

Application of Taxonomy to Managing, Managed Environment and Additional Functions

	Managing Environment	Managed Environment	Additional Functions
Business Management:			
Inventory Control	Yes	Yes	Yes
Accounting	Yes	Yes	Yes
Policy Administration			Yes
Business Strategic Planning			Yes
Process Management			Yes
Information Services Management			Yes
Organizational Planning			Yes
Configuration Management:			
Configuration Design	Yes		Yes
Environmental Planning			Yes
Configuration Creation	Yes		Yes
Updating Configuration	Yes	Yes	
Accessing Configuration	Yes	Yes	
Software Administration:			
Licensing	Yes	Yes	Yes
Planning			Yes
Distribution	Yes	Yes	
Synchronization	Yes	Yes	
Installation		Yes	
Activation	Yes	Yes	
Testing		Yes	
Backout		Yes	
Monitoring and Tracking	Yes		
Operations Management:			
Workload and Operations Planning			Yes
Workload Control	Yes	Yes	
Operations Control	Yes	Yes	
Print Management	Yes		
Performance Management:			
Performance Planning			Yes
Performance Control and Monitoring	Yes		
Performance Execution and Measurement		Yes	
Problem Management:			
Problem Process Planning	Yes		Yes
Problem Policy Planning	Yes	Yes	
Problem Determination	Yes		
Problem Analysis	Yes	Yes	
Problem Bypass and Recovery	Yes	Yes	
Problem Assignment	Yes		
Problem Resolution and Verification	Yes	Yes	
Security Management:			
Authentication	Yes	Yes	
Access Control	Yes	Yes	
Non-repudiation		Yes	
Integrity	Yes	Yes	
Confidentiality	Yes	Yes	
Security Audit	Yes	Yes	Yes
Key Management	Yes		

Data Size Neutrality and 64-bit Support

Andrew Josey <a.josey@opengroup.org> *of The Open Group reports on changes in the Single UNIX Specification, Version 2.*

The Single UNIX Specification, Version 2, provides enhanced support for 64-bit programming models by being n-bit clean and data size neutral. This article is a brief introduction to 64-bit programming models, data size neutrality, and application porting issues.

Introduction

When the UNIX operating system was first created in 1969 it was developed to run on a 16-bit computer architecture. The C language of the time supported 16-bit integer and pointer data types and also supported a 32-bit integer data type that could be emulated on hardware that did not support 32-bit arithmetic operations.

When 32-bit computer architectures, which supported 32-bit integer arithmetic operators and 32-bit pointers, were introduced in the late 1970s, the UNIX operating system was quickly ported to this new class of hardware platforms. The C language data model developed to support these 32-bit architectures quickly evolved to consist of a 16-bit short-integer type, a 32-bit integer type, and a 32-bit pointer. During the 1980s, this was the predominant C data model available for 32-bit UNIX platforms.

To describe these two data models in modern terms, the 16-bit UNIX platforms used an IP16 data model, while 32-bit UNIX platforms use the ILP32 programming model. The notation describes the width assigned to the basic data types; for example, ILP32 denotes that **int** (I), **long** (L), and **pointer** (P) types are all 32-bit entities. This notation is used extensively throughout this article.

The first UNIX standardization effort was begun in 1983 by a /usr/group committee. This work was merged into the work program of the IEEE POSIX committees in 1985. By 1988, both POSIX and X/Open committees had developed detailed standards and specifications that were based upon the predominant UNIX implementations of the time. These committees endeavored to develop architecture-neutral definitions that could be implemented on any hardware architecture.

The transition from 16-bit to 32-bit processor architectures happened quite rapidly just before the UNIX standardization work was begun. Since the specifications were based on existing practice and the predominant data model did not change during this gestation period, some dependencies upon the ILP32 data model were inadvertently incorporated into the final specifications.

Most of today's 32-bit UNIX platforms use the ILP32 data model. However another data model, the LP32 model, is also very popular for other operating systems. The majority of C-language programs written for Microsoft Windows 3.1 are written for the Win-16 API which uses the LP32 data model. The Apple Macintosh also uses the LP32 data model.

32-bit platforms have a number of limitations that are increasingly a source of frustration to developers of large applications, such as databases, who wish to take advantage of advances in computer hardware. There is much discussion today in the computer industry about the barrier presented by 32-bit addresses. 32-bit pointers can address only 4GB of virtual address space. There are ways of overcoming this limitation, but application development is more complicated and performance is significantly reduced. Until recently the size of a data file could not exceed 4GB. However, the 4GB file size limitation was overcome by the Large File Summit extensions which are included in XSH, Issue 5.

Disk storage has been improving in real density at the rate of 70% compounded annually, and drives of 4GB and larger are readily available. Memory prices have not dropped as sharply, but 16MB chips are readily available, with 64MB chips in active development. CPU processing power continues to increase by about 50% every 18 months, providing the power to process ever larger quantities of data. This conjunction of technological forces, along with the continued demand for systems capable of supporting ever-larger databases, simulation models and full-motion video, have generated requirements for support of larger addressing structures.

A number of 64-bit processors are now available, and the transition from 32-bit to 64-bit architectures is rapidly occurring among all the major hardware vendors. 64-bit UNIX platforms do not suffer from the file size or flat address space limitations of 32-bit platforms. Applications can access files that occupy terabytes of disk space because 64-bit file offsets are possible. Similarly, applications can now theoretically access terabytes of memory because pointers can be 64 bits. More physical memory results in faster operations. The performance of memory-mapped file access, caching, and swapping, is greatly improved. 64-bit virtual addresses simplify the design of large applications. All the major database vendors now support 64-bit platforms because of dramatically improved performance for very large database applications available on very large memory (VLM) systems.

The world is currently dominated by 32-bit computers, a situation that is likely to continue for the near future. These computers run 16- or 32-bit applications or some mixture of the two. Meanwhile, 64-bit computers will run 32-bit code, 64-bit code, or mixtures of the two (and perhaps even some 16-bit code). New 64-bit applications and operating systems must integrate smoothly into this environment. Key issues facing the computing industry are the interchange of data between 64- and 32-bit systems (in some

cases on the same system) and the cost of maintaining software in both environments. Such interchange is especially needed for large application suites such as database systems, where one may want to distribute most of the applications as 32-bit binaries that run across a large installed base, but be able to choose 64-bits for a few crucial applications.

64-bit Data Models

Prior to the introduction of 64-bit platforms, it was generally believed that the introduction of 64-bit UNIX operating systems would naturally use the ILP64 data model. However, this view was too simplistic and overlooked optimizations that could be obtained by choosing a different data model.

Unfortunately, the C programming language does not provide a mechanism for adding new fundamental data types. Thus, providing 64-bit addressing and integer arithmetic capabilities involves changing the bindings or mappings of the existing data types or adding new data types to the language.

ISO/IEC 9899:1990, Programming Languages - C (ISO C) left the definition of the **short int**, the int, the **long int**, and the **pointer** deliberately vague to avoid artificially constraining hardware architectures that might benefit from defining these data types independently from one another. The only constraints were that **int**s must be no smaller than **short**s, and **long**s must be no smaller than **int**s, and **size_t** must represent the largest unsigned type supported by an implementation. It is possible, for instance, to define a **short** as 16 bits, an **int** as 32 bits, a **long** as 64 bits and a **pointer** as 128 bits. The relationship between the fundamental data types can be expressed as:

```
sizeof(char) <= sizeof(short) <= sizeof(int)
        <= sizeof(long) = sizeof(size_t)
```

Ignoring non-standard types, all three of the following 64-bit pointer data models satisfy the above relationship:

- LP64 (also known as 4/8/8)
- ILP64 (also known as 8/8/8)
- LLP64 (also known as 4/4/8).

The differences between the three models lies in the non-pointer data types. The table below details the data types for the above three data models and includes LP32 and ILP32 for comparison purposes.

Data Type	LP32	ILP32	ILP64	LLP64	LP64
char	8	8	8	8	8
short	16	16	16	16	16
int32			32		
int	16	32	64	32	32
long	32	32	64	32	64
long long (int64)					64
pointer	32	32	64	64	64

When the width of one or more of the C data types is changed, applications may be affected in various ways. These effects fall into two main categories:

- Data objects, such as a structure, defined with one of the 64-bit data types will be different in size from those declared in an identical way on a 16 or 32-bit system.
- Common assumptions about the relationships between the fundamental data types may no longer be valid in a 64-bit data model. Applications which depend on those relationships often cease to work properly when compiled on a 64-bit platform. A typical assumption made by many application developers is that:
  sizeof(int) = sizeof(long) = sizeof(pointer)
  This relationship is not codified in any C programming standard, but it is valid for the ILP32 data model. However, it is not valid for two of the three 64-bit data models described above, nor is it valid for the LP32 data model.

The ILP64 data model attempts to maintain the relationship between **int**, **long**, and **pointer** which exists in the ILP32 model by making all three types the same size. Assignment of a **pointer** to an **int** or a **long** will not result in data loss.

The downside of this model is that it depends on the addition of a new 32-bit data type such as **int32** to handle true 32-bit quantities. There is thus a potential for conflict with existing **typedefs** in applications. An application which was developed on an ILP32 platform, and subsequently ported to an ILP64 platform, may be forced to make frequent use of the **int32** data type to preserve the size and alignment of data because of interoperability requirements or binary compatibility with existing data files.

The LLP64 data model preserves the relationship between **int** and **long** by leaving both as 32-bit types. Data objects, such as structures, which do not contain pointers will be the same size as on a 32-bit system. This model is sometimes described as a 32-bit model with 64-bit addresses. Most of the run-time problems associated with the assumptions between the sizes of the data types are related to the assumption that a pointer will fit in an **int**. To solve this class of problems, **int** or long variables which should be 64 bits in length are changed to **long long** (or **int64**), a non-standard data type. This data model is thus again dependent on the introduction of a new data type. Again there is potential for conflict with existing **typedefs** in applications.

The LP64 data model takes the middle road. 8, 16, and 32-bit scalar types (.B **char** , **short**, and **int**) are provided to support objects that must maintain size and alignment with 32-bit systems. A 64-bit type, **long**, is provided to support the full arithmetic capabilities, and is available to use in conjunction with pointer arithmetic. Applications that assign addresses to scalar objects need to specify the object as **long** instead of **int**.

In the LP64 data model, data types are natural. Each scalar type is larger than the preceding type. No new data types are required. As a language design issue, the purpose of having **long** in the language

anticipates cases where there is an integral type longer than int. The fact that int and long represent different width data types is a natural and common sense approach, and is the standard in the PC world where int is 16-bits and long is 32-bits.

A major test for any C data model is its ability to support the very large existing UNIX applications code base. The investment in code, experience, and data surrounding these applications is the largest determiner of the rate at which new technology is adopted and spread. In addition, it must be easy for an application developer to build code that can be used in both existing and new environments.

The UNIX development community is driven technically by a set of API agreements embodied in standards and specifications documents from groups such as X/Open, IEEE, ANSI, and ISO. These documents were developed over many years to codify existing practice and define agreement on new capabilities. As a result these specifications are of major value to the system developers, application developers, and end-users. There are numerous test suites that verify that implementations correctly embody the details of a specification and certify that fact to interested parties. Any 64-bit data model cannot invalidate large portions of these specifications and expect to achieve wide adoption.

A number of vendors have extensive experience with the LP64 data model. By far, the largest body of existing 32-bit code already modified for 64-bit environments runs on LP64 platforms. Experience has shown that it is relatively easy to modify existing code so that it can be compiled on either an 32-bit or 64-bit platform. Interoperability with existing ILP32 platforms is well proven and is not an issue. At least one LP64-based operating system (Digital UNIX V4.0) has met and passed the majority of existing verification suites and has obtained the UNIX 95 brand.

A small number of ILP64-based platforms have also shipped. These have demonstrated that it is feasible to complete the implementation of an ILP64 environment. However, as of early 1997, no LLP64 or ILP64-based systems had achieved the same level of standards conformance or met the requirements of the UNIX 95 brand.

Although the number of applications written in C requiring a large virtual address space is growing rapidly, there has not been a requirement to date for a 64-bit int data type. The majority of existing 64-bit applications previously ran only on 32-bit platforms, and had no expectation of a greater range for the int data type. The extra 32 bits of data space in a 64-bit int would appear to be wasted. Any future applications that require a larger scalar data type can use the long type.

Nearly all applications moving from a 32-bit platform require some minor modifications to handle 64-bit pointers, especially where erroneous assumptions about the relative size of int and pointer data types were made. Common assumptions about the relative sizes of int, char, short, and float data types generally do not cause problems on LP64 platforms (since the sizes of those data types are identical to those on an ILP32 platform), but do so on an ILP64 platform.

Other language implementations will continue to support a 32-bit int type. For example, the FORTRAN-77 standard requires that the type INTEGER be the same size as REAL, which is half the size of DOUBLE PRECISION. This, together with customer expectations, means that FORTRAN-77 implementations will generally leave INTEGER as a 32-bit type, even on 64-bit platforms. A significant number of applications use C and FORTRAN together, either calling each other or sharing data files. Such applications have been among the first to move to 64-bit environments. Experience has shown that it is usually easier to modify the data sizes and types on the C side than the FORTRAN side of such applications. These applications will continue to require a 32-bit int data type in C regardless of the size of the int data type.

In 1995, a number of major UNIX vendors agreed to standardize on the LP64 data model for a number of reasons:

- Experience suggests that neither the LP64 nor the ILP64 data models provide a painless porting path from a 32-bit platform, but that all other things being equal, the smaller data types in the LP64 data model enable better application performance.
- A crucial investment for end-users is the existing data built up over decades in their computer systems. Any proposed solution must make it easy to utilize such data on a continuing basis. Unfortunately, the ILP64 data model does not provide a natural way to describe 32-bit data types, and must resort to non-portable constructs such as int32 to describe such types. This is likely to cause practical problems in producing code which can run on both 32- and 64-bit platforms without numerous #ifdef constructions. It has been possible to port large quantities of code to LP64 platforms without the need to make such changes, while maintaining the investment made in data sets, even in cases where the typing information was not made externally visible by the application.
- Most ints in existing applications can remain as 32 bits in a 64-bit environment; only a small number are expected to be the same size as pointer or long. Under the ILP64 data model, most ints will need to change to int32. However, int32 does not behave like a 32-bit int. Instead, int32 is like short in that all operations have to be converted

to int (64-bits, sign extended) and performed in 64-bit arithmetic. Thus, int32 in the ILP64 data model is not exactly the same as int in the ILP32 data model. These differences may cause subtle and hard-to-find bugs.

- Instruction cycle penalties are incurred whenever additional cycles are required to properly implement the semantics of the intended data model. For example, in the LP64 data model it is only necessary to perform sign extension on int when you have a mixed expression including longs. However, most integral expressions do not include longs and compilers can be made smart enough to only sign extend when necessary.
- int is by far the most frequent data type to be found (statically and sometimes dynamically) within C and C++ programs. 64-bit integers require twice as much space as 32-bit integers. Applications using 64-bit integers consume additional memory and CPU cycles transporting that memory throughout the system. Furthermore, the latency penalty of 64-bit integers can be enormous, especially to disk, where it can exceed 1,000,000 CPU cycles (3 nsec to 3 msec). The memory size penalty for unneeded 64-bit integers could therefore be very high for some applications.
- The LP64 data model enhances portability, especially for combined FORTRAN and C applications, and the most common types of problems that can occur are susceptible to automatic detection.
- Interoperability is improved by the ability to use a standard data type to declare data structures that can be used in both 32-bit and 64-bit environments.
- Standards conformance has been demonstrated both in the practical sense by the porting of many programs and in the formal sense of compliance with industry standards through verification test suites.
- Transition from the current industry practice is smooth and direct following a path grooved with experience and demonstrated success.
- No new non-portable data types are required. The data model makes natural use of the C fundamental data types.

Data Size Neutrality

When it was understood that the Single UNIX Specification was constraining system implementations that were other than ILP32, the relevant specifications were reviewed and recommendations drafted to make these specifications data size- and architecture-neutral. These recommendations were incorporated into the Single UNIX Specification, Version 2 published in 1997.

Porting Issues

Porting an application to a 64-bit UNIX system can be accomplished with a minimal amount of effort if the application was developed using good modern software engineering practices such as:

- ISO C function prototypes
- consistent and careful use of data types
- all declarations are in headers

First of all, determine which data model is available on the platform to which you are porting. This data model will have a major impact on the amount of work required to achieve a successful port.

Then take the time to create and use ISO C function prototypes if they are absent from the source code. Unfortunately large quantities of perfectly good legacy code developed in the days before portability was a major issue may not have function prototypes. Fortunately many compilers have an option to generate ISO C function prototypes.

The remainder of this article assumes that you are porting to an LP64 platform since this is the data model of choice amongst major vendors, but the issues raised are equally valid on some or all of the other 64-bit data models.

General

Use utilities such as `grep` to locate and check all instances of the following:

- Shift and complement operators; that is, "<<", ">>", "~". If used with long, add "L" to value shifted to avoid an incorrect result.
- Addresses of objects ("&") should not be stored in an int.
- Declarations of type long. Many of these can be converted to type int to save space. This is particularly true for network code.
- The functions `lseek()`, `fseek()`, `ftell()`, `fgetpos()`, and so on. Use either off_t or fpos_t as appropriate for offset arguments. Do not use int or long to store file offsets.
- All (int *) and (long *) casts.
- Use of (char *) 0 for zero or (char *) comparisons. Use NULL instead.
- Hard-coded byte counts or memory sizes. These will be wrong if they assume longs or pointers are 32 bits. Applications should use the `sizeof()` operator to avoid such problems.

Declarations

To enable application code to work on both 32-bit and 64-bit platforms, check all int and long declarations. Declare integer constants using "L" or "U" as appropriate. Ensure an unsigned int is used where appropriate to prevent sign extension. If you have specific variables that need to be 32 bits on both platforms, define the type to be int. If the variable should be 32 bits on an ILP32 platform and 64 bits on an LP64 platform, define the variables to be long.

Declare numeric variables as int or long for alignment and performance. Don't

worry about trying to save bytes by using char or short. Remember that if the type specifier is missing from a declaration, it defaults to an int. Declare character pointers and character bytes as unsigned to avoid sign extension problems with 8-bit characters.

Assignments and Function Parameters

All assignments require checking. Since pointer, int, and long are no longer the same size on LP64 platforms, problems may arise depending on how the variables are assigned and used within an application.

Do not use int and long interchangeably because of the possible truncation of significant digits, as shown in the following example:

```
int iv;
long lv;
iv = lv;
```

Do not use int to store a pointer. The following example works on an ILP32 platform but fails on an LP64 platform because a 32-bit integer cannot hold a 64-bit pointer:

```
unsigned int i, *p;
i = (unsigned) p;
```

The converse of the above example is sign extension:

```
int *p;
int i;
p = (int *)i;
```

Do not pass long arguments to functions expecting int arguments. Avoid assignments similar to the following:

```
int foo(int);
int iv;
long lv;
iv = foo( lv );
```

Do not freely exchange pointers and ints. Assigning a pointer to an int, assigning back to a pointer, and dereferencing the pointer may result in a segmentation fault. Avoid assignments similar to the following example:

```
int iv;
char *buffer;
buffer = (char *) malloc
          ((size_t)MAX_LINE);
iv = (int) buffer;
...
buffer = (char *) iv;
```

Do not pass a pointer to a function expecting an int as this will result in lost information. For example, avoid assignments similar to the following:

```
void f();
char *cp;
f(cp);
```

Use of ISO C function prototypes should avoid this problem. Use the void* type if you need to use a generic pointer type. This is preferable to converting a pointer to type long.

Examine all assignments of a long to a double as this can result in a loss of accuracy. On an ILP32 platform, an application can assume that a double contains an exact representation of any value stored in a long (or a pointer). On LP64 platforms this is no longer a valid assumption.

External Interfaces

An external interface mismatch occurs when an external interface requires data in a particular size or layout, but the data is not supplied in the correct format.

For example, an external interface may expect a 64-bit quantity, but receive instead a 32-bit quantity. Another example is an external structure which expects a pointer to a structure with 2 ints (8 bytes) but instead receives a pointer to a structure with an int and a long (16 bytes, 12 of data, 4 of alignment padding). External interface mismatching is a major cause of porting problems.

Format Strings

The function `printf()` and related functions can be a major source of problems. For example, on 32-bit platforms, using "%d" to print either an int or long will usually work, but on LP64 platforms "%ld" must be used to print a long. Use the modifier "l" with the d, u, o, and x conversion characters to specify assignment of type long or unsigned long. When printing a pointer, use "%p". If you wish to print the pointer as a specific representation, the pointer should be cast to an appropriate integer type before using the desired format specifier. For example, to print a pointer as a unsigned long decimal number, use %lu:

```
char *p;
printf( "%p %lu\n", (void *)p,
          (unsigned long)p );
```

As a rule, to print an integer of arbitrary size, cast the integer to long or unsigned long and use the "%ld" conversion character.

Constants

The results of arithmetic operations on a 64-bit platform can differ from those obtained using the same code on a 32-bit platform. Differing results are often caused by sign extension problems. These are generally the result of mixing signed and unsigned types and the use of hexadecimal constants. Consider the following code example:

```
long lv = 0xFFFFFFFF;
if ( lv < 0 ) {
```

On an ILP32 platform, `lv` is interpreted as -1 and the `if` condition succeeds. On an LP64 platform `lv` is interpreted as 4294967295 and the `if` condition fails.

Pointers

On ILP32 platforms, an int and a pointer are the same size (32 bits) and application code can generally use them interchangeably. For example, a structure could contain a field declared as an int, and most of the time contain an integer, but occasionally be used to store a pointer.

Another example, which most 32-bit int utilities will not catch, is the following:

```
int iv, *pv;
iv = (int) pv;
pv = (int *) iv;
```

This code fails on an LP64 platform. Not only do you lose the high 4 bytes of "p", but by default these high bytes are significant.

Sizeof()

On ILP32 platforms **sizeof(int) = sizeof(long) = sizeof(ptr *)**. Using the wrong `sizeof()` operand does not cause a problem. On LP64 platforms, however, using the wrong `sizeof()` will almost certainly cause a problem. For example, the following 32-bit code which copies an array of **pointers** to **ints**:

```
memcpy((char *)dest, (char
   *)src, number * sizeof(int))
```

must be changed to use **sizeof(int *)**:

```
memcpy((char *)dest, (char
*)src, number * sizeof(int *))
```

on an LP64 platform.

Note that the result of the `sizeof()` operation is type **size_t** which is an unsigned **long** on LP64 platforms.

Structures and Unions

The size of structures and unions on 64-bit platforms can be different from those on 32-bit platforms. For example, on ILP32 platforms the size of the following structure is 8 bytes:

```
struct Node {
  struct Node *left;
  struct Node *right;
}
```

but on an LP64 platform its size is 16 bytes.

If you are sharing data defined in structures between 32-bit and 64-bit platforms, be careful about using **longs** and **pointers** as members of shared structures. These data types introduce sizes that are not generally available on 32-bit platforms. Avoid storing structures with pointers in data files. This code then becomes non-portable between 32-bit and 64-bit platforms.

To increase the portability of your code, use **typedef**'d types for the fields in structures to set up the types as appropriate for the platform, and use the `sizeof()` operator to determine the size of a structure. If necessary, use the **#pragma** pack statement to avoid compiler structure padding (This is not portable and is not a general solution). This is important if data alignment cannot change (network packets, and so on).

Structures are aligned according to the strictest aligned member. Padding may be added to ensure proper alignment. This padding may be added within the structure, or at the end of the structure to terminate the structure on the same alignment boundary which it started.

Problems can occur when the use of a union is based on an implicit assumption, such as the size of member types.

Consider the following code fragment which works on ILP32 platforms. The code assumes that an array of two **unsigned long** overlays a double.

```
union double_union {
  double d;
  unsigned long ul[2];
};
```

To work on an LP64 platform, `ul` must be changed to an **unsigned int** type:

```
union double_union {
  double d;
  unsigned int ul[2]
};
```

This problem also occurs when building unions between **ints** and **pointers** since they are not the same size on LP64 platforms.

Beware of all aliasing of different multiple definitions of the same data. For example, assume the following two structures refer to the same data in different ways:

```
struct node {
  int src_addr, dst_addr;
  char *name;
}
struct node {
  struct node *src, *dst;
  char *name;
}
```

This works on an ILP32 platform, but fails on an LP64 platform. The two structure definitions should be replaced with a union declaration to ensure portability.

More Information

This article is derived from The Open Group Source Book, *Go Solo 2 – The Authorized Guide to Version 2 of the Single UNIX Specification*. This is published herein with permission of The Open Group. More information on the Single UNIX Specification, Version 2, can be obtained from the following sources:

The online version of the Single UNIX Specification can be found at: <http://www.opengroup.org/unix/online.html>.

The Open Group Source Book *Go Solo 2 – The Authorized Guide to Version 2 of the Single UNIX Specification*, 600 pages, ISBN 0-13-575689-8. This book provides complete information on what's new in Version 2, with technical papers written by members of the working groups that developed the specifications , and a CD-ROM containing the complete 3000 page specification in both HTML and PDF formats (including PDF reader software). For more information on the book, see <http://www.opengroup.org/unix/gosolo2>.

Additional information on the Single UNIX Specification can be obtained at The Open Group Web site, <http://www.opengroup.org/unix/>.

nternet Practices

im Isaak <Jim.Isaak@digital.com> reports on he October Meeting in Reno, Nevada

The Internet Practices Study Group met n Reno with the Portable Application tandards Committee (PASC) groups. his meeting agreed to move forward a roject Authorization Request (PAR) for ntranet/Extranet best practices, and to urther research practices in the area of echnical publishing.

he first meeting of the working group or this new project will be January 2–13, 1998 at the PASC meetings in Ft. auderdale. Copies of minutes, PAR, and elated materials can be found at: http://computer.org/standard/Internet/>.

his work will be of interest to webmas-rs, particularly those responsible for orporate intranet and extranet opera-ons, as a way to gain from the experi-nces of others, and also to establish redibility and confidence in Web-based perations. In some areas, such as protec-on of property rights (copyright and roprietary information), best practices an provide an essential line of "defense" or offense) in liability and/or legal sputes.

sues to be considered include copyright, roprietary data declarations, indexing nd content classification of pages, use of och transparent dates, context (author, sponsible organization, currency, etc.), ultinational sensitivities, browser toler-nce, accommodation of persons with sabilities, and bandwidth efficiencies. he objective is to improve the produc-ity of Intra/Extranet Web operations in rms of locating relevant information, d efficient development and mainte-nce practices.

This will also be of interest to creators of tools, such as dynamic page generation, or page development/design tools. Those products will be able to claim that they generate Web pages consistent with IEEE standards for best practices, and for indexing and other services where coherent use of Metadata and other Web page elements can be of significant value.

It is expected that balloting on a first version of this document will start in June of 1998. Since a "comprehensive" set of recommendations is not likely in the short term, the objective is to nail down a priority set, and bring these to ballot while working on an additional set concurrently.

the bookworm

Books reviewed in this column:

Douglas E. Comer

Computer Networks and Internets

Upper Saddle River, NJ: Prentice Hall, 1997.
ISBN 0-13-239070-1. Pp. 506 + CD-ROM by Ralph Droms.

Douglas E. Comer

The Internet Book

2nd ed. Upper Saddle River, NJ: Prentice Hall, 1997.
ISBN 0-13-890161-9. Pp. 327.

Douglas E. Comer & David L. Stevens

Internetworking with TCP/IP

vol. 3: "Client-Server Programming...; Windows Sockets Version." Upper Saddle River, NJ: Prentice Hall, 1997.
ISBN 0-13-848714-6. Pp. 513.

V.S. Dasan & L.R. Ordorica

Hands-On Intranets

Upper Saddle River, NJ: Prentice Hall, 1998.
ISBN 0-13-857608-4. Pp. 326.

Bil Lewis & Daniel J. Berg

Multithreaded Programming with PThreads

Upper Saddle River, NJ: Prentice Hall, 1997.
ISBN 0-13-443698-9. Pp. 368.

Jerry Peek, et al.

UNIX Power Tools

2nd ed. Sebastopol, CA: O'Reilly & Associates, 1997.
ISBN 1-56592-260-3. Pp. 1073 + CD-ROM.

Steve Oualline

Practical C Programming

3rd ed. Sebastopol, CA: O'Reilly & Associates, 1997.
ISBN 1-56592-306-5. Pp. 428.

Graham Hamilton, et al.

JDBC Database Access with Java

Reading, MA: Addison Wesley Longman, 1997.
ISBN 0-201-30995-5. Pp. 462 + ref. card.

David A. Taylor

Object Technology: A Manager's Guide

2nd ed. Reading, MA: Addison Wesley Longman, 1997.
ISBN 0-201-30994-7. Pp. 205.

by Peter H. Salus

Peter H. Salus is a member of ACM, the Early English Text Society, the Trollope Society, and is a life member of the American Oriental Society. He has held no regular job in the past lustrum. He owns neither a dog nor a cat.

<peter@pedant.com>

I'm writing this at the beginning of October. You'll be reading it after Thanksgiving (yes, I know that Canadian Thanksgiving is earlier and that the rest of the world doesn't celebrate it at all). This gives everyone some time to get spouses, significant others, parents, or whatever to buy them the books on this year's list before Christmas or New Year's. The list is at the end of the column.

The UPS strike hit book distribution hard. My average number of packages dropped from five to two per day. I had only about 120 books to select from, and only a half-dozen were Java. Happily, Tcl/Tk seems to have (finally) caught the attention of some of the publishers, and several good books have appeared (three of them reviewed last issue).

A Comer Trifecta

A new book by Doug Comer is something that grabs anyone interested in networking, whether a small Ethernet or the Matrix. *Computer Networks and Internets* is an extraordinary piece of work. It not only supplies information on how networks operate; it does so in Comer's literate and readable style, avoiding excessive detail. The accompanying CD-ROM, by Ralph Droms, contains a lot of useful stuff: animations, figures, images, data, each presented as both a presentation file and a basic file.

Next, there's a new edition of Comer's *Internet Book*. I liked the first edition in 1995; the second edition is not merely better; it is only 25 pages longer. It's the perfect book for the manager, director, or vice president who just isn't aware of what's happening. The only thing I dislike is the lack of any sort of references, further reading, etc.

Third, there's a Winsock version of volume 3 of Comer and Stevens, *Internetworking with TCP/IP*. It begins well, as do the AT&T and TLI and BSD Sockets versions. I can't really appraise the Winsock stuff, because I've got no machine running Win95 or NT. If you are so afflicted, I'm sure this is for you.

Intranets

I've gotten about half a dozen books with Intranet in the title and nearly as many with Extranet. Dasan and Ordorica have created a book that's worthwhile, as opposed to most of the others. I was especially taken by their matter-of-fact attitude and their "further reading" (appendix A).

Tools and Threads

Multithreaded Programming with PThreads is a really fine introduction to threads. Interestingly, P1003.1c has been implemented on all the UNIX platforms, Linux, and VMS. Win32 threads under NT and OS/2 threads are noncompliant. There seem to be no plans in Redmond, WA, to adapt to PThreads, but there are freeware POSIX libraries. OS/2 has an optional POSIX library. This is a well-written, handy book that teaches you quicker than a greased Kolstad (circa 1980).

The second edition of *UNIX Power Tools* is gigantic, thorough, and comes with a neat CD-ROM. But I found two drawbacks: first, the authors still haven't discovered zsh; and, second, the typesetting is such that items that I think need emphasis are grayed down. Some (not much) of my book is unintelligible because I can't read parts of sentences. For some books I get, this would be a feature. For an O'Reilly, there's only one word: weird.

liked *Practical C Programming* in 1991. he 1993 edition converted everything to NSI C. Whether you're doing some- ing sensible or using Microsoft C++, is continues to be a very useful book in s third edition.

ave a Cuppa

he Java Database Connectivity API is art of JDK 1.1. Now we have a "tutorial nd annotated reference" for JDBC writ- n by its developers. It's authoritative nd useful. I especially liked the sections n how JDBC fits in to SQL and rdbs in eneral.

et More Objects

he first edition of Taylor's *Object echnology* appeared in 1990. I recom- ended it at that time to all those whose anagers needed education in just what would mean to use "objects." It's still ue in the second edition. Smalltalk, ++, and Java are discussed; there is a ention of Eiffel and Ada95, but none of odula3 or CLOS. Tch, tch.

ne Bookworm's Top Ten for 1997

ot ranked in order)

Donald A. Knuth, *The Art of Computer Programming*, vol. 1, 3rd. ed. (Addison Wesley Longman)

Ralph E. Griswold & Madge T. Griswold, *The Icon Programming Language* (Peer-to-Peer Communications)

Joe Duran & Charlie Sauer, *Mainstream Videoconferencing* (Addison Wesley Longman)

Bjarne Stroustrup, *The C++ Programming Language*, 3rd ed. (Addison Wesley Longman)

5–7. Security

Rubin, Geer & Ranum, *Web Security Sourcebook* (John Wiley & Sons)
Garfinkel & Spafford, *Web Security & Commerce* (O'Reilly & Associates)
Ira Winkler, *Corporate Espionage* (Prima)

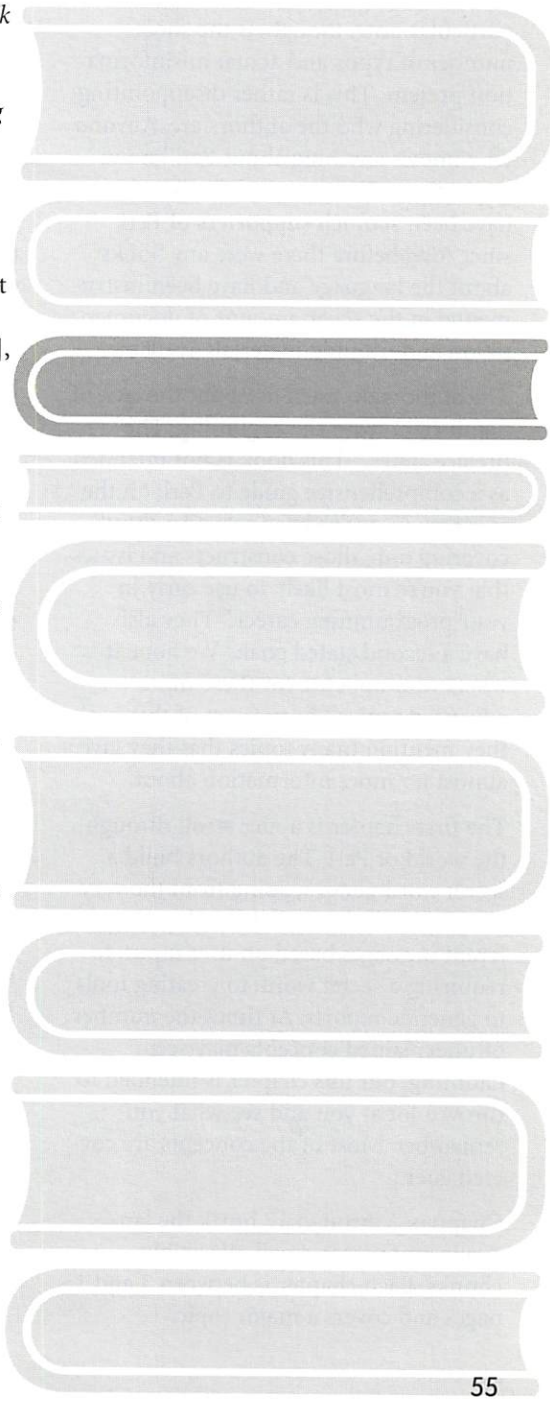
8-10. Tcl/Tk

Harrison & McLennan, *Effective Tcl/Tk Programming* (Addison Wesley Longman)
Brent B. Welch, *Practical Programming in Tcl and Tk*, 2nd ed. (Prentice Hall)
Mark Harrison, et al., *Tcl/Tk Tools* (O'Reilly & Associates)

(If 13 is a baker's dozen, what's 11? Following Lou Katz's old theory, 10 must be a metric dozen [prices don't go "up"–you just get ten for the price of 12], so 11 must be a metric baker's dozen. Send me your ideas, and I will print some of them. We seem to have solved the problem with C++, as have the French with "bis," but not colloquially in English. All of this is prelude to the eleventh book on the list of ten. It was reviewed last December, but was too late for last year's list.)

I still have nothing but admiration for Abelson, Sussman, and Sussman, *Structure and Interpretation of Computer Programs* (MIT Press).

I'd like to thank *UNIX Review* for naming *Lion's Commentary on UNIX* 6th ed. (Peer-to-Peer) its book-of-the-year. John and his family were touched by this belated recognition. There are a lot o' books in them thar stores. Have a happy holiday and read lots of them.



book reviews

Randal Schwartz and Tom Christiansen

Learning Perl, Second Edition

O'Reilly & Associates, 1977. ISBN 1-56592-284-0.
Pp. 269. $29.95.

Reviewed by Stephen Potter

<spp@psa.pencom.com>

Learning Perl is an adequate, if sometimes shallow, introduction to Perl. Of particular note, though, is the sheer number of typos and actual misinformation present. This is rather disappointing, considering who the authors are. Anyone who knows anything about Perl knows the authors. Schwartz and Christiansen have been staunch supporters of Perl since long before there were any books about the language and have been instrumental in the sheer amount of documentation and training materials available.

The authors do warn us about the lack of depth right from the beginning. The preface states, "This book is not intended as a comprehensive guide to Perl; on the contrary . . . we've been selective about covering only those constructs and issues that you're most likely to use early in your programming career." They also have a second stated goal: "We hope it whets your appetite for these more advanced topics." In support of this goal, they mention many topics that they give almost no more information about.

The first chapter is a nice stroll through the world of Perl. The authors build a secret word guesser, going from the very simple "Hello, World" through giving different messages based on user input, to requiring a secret word, to creating tools to generate reports. At times the number of unexplained concepts may seem daunting, but this chapter is intended to throw a lot at you and see what you remember. Most of the concepts are covered later.

Chapters 2 through 17 break the language syntax into small, digestible chunks. Each chapter is between 3 and 15 pages and covers a major topic.

Unfortunately, some of the information is thrown in rather haphazardly. For example, some of the last parts of chapter 2 ("Scalar Data") are "<STDIN> as a Scalar Value" and "Output with Print," but later an "entire" chapter (chapter 6, "Basic I/O") is devoted to this subject with "Input from STDIN," "Input from the Diamond Operator," and "Output to STDOUT."

Chapter lengths seem rather strange as well. Chapter 2 is 15 pages long. Chapter 3 ("Arrays and List Data") is only 9 pages. Chapter 5 ("Hashes"), one of the most important topics for programming the Perl way, is only 4.

Bowing to the recent marketing hype of the World Wide Web, chapter 19 ("CGI Programming") is one of the longest chapters in the book at 29 pages. On the positive side, a lot of advanced concepts are introduced in this chapter. On the negative side, they aren't well discussed, being that they are advanced, and this book is an introduction.

The most disturbing problem with *Learning Perl* is the typos, misinformation, and continuity breaks. O'Reilly generally produces consistently well written and edited books. Unfortunately, the authors' reputations seem to have caused a lower-than-normal level of attention to be given to this work. For example, in the third chapter, the authors variously tell us, "If you access an array element ... an index of less than zero . . . the 'undef' value is returned" and "A negative subscript on an array counts back from the end. So, another way to get at the last element is with the subscript -1."

Another prime example of the continuity problems is a random quote from page 144: "Despite its other shortcomings, the 'local' operator can do one thing that 'my' cannot: it can give just one element of an array or a hash a temporary value." This would be interesting if we had any idea what the shortcomings of "local" were. In fact, based on the only discussion of "local" and "my" in the book (pages 96–99), it would seem that "local" is a much better operator than "my."

Most of the typos and misinformation should be fixed by the second or third printing. Before you buy a copy of *Learning Perl, Second Edition*, check the printing history. Look for a line like "September 1997: Minor Corrections." If you don't see this, check page 1 of chapter. If the second paragraph (which starts "After playing with this version") contains the typo "asking for ways to do this that,f or the other," pass on this copy of the book. If you have a good grounding in UNIX, shell scripting, or C programming, you can muddle your way through this book without too many problems. Just remember, if you see something that doesn't look right to you, try it yourself. Perl is a wonderfully rich language, filled with many interesting ways of doing things. Some of the things that don't look right may well be right. Some of them may simply be typos in the book.

Michael and Ronda Hauben

Netizens: On the History and Impact of Usenet and the Internet

IEEE Computer Society, 1997. ISBN 0-8186-7706-
Pp. 345. $28.95.

Reviewed by Daniel Lazenby

<dlazenby@ix.netcom.com>

The title says it all. This book tells the story of how ordinary people have mad and can make a difference. Often the re olution caused by a technology and the people who quietly nurtured and foster it into being is not recorded until well after the fact. *Netizens* strives to capture the history while some founders are stil able to provide firsthand accounts. This easily read book chronicles the evolutio of USENET and the Internet. Not only does *Netizens* chronicle the past; it striv to illustrate the life-changing influence USENET and the Internet have had on people and society. The book also takes

:w moments to ponder the changes yet) come. This book is based on academic :search papers that Michael and Ronda riginally published on the Internet.

'etizens is broken into four major parts, The Present," "The Past," "And the ıture," and "Contributions Toward eveloping a Theoretical Framework." he first part recaps what has been creat- l and how it was created. "The Past" views where USENET and the Internet ıme from. This part of the book :plores the grassroots beginnings of SENET and the gestation of what is)w known as the Internet. The third ırt explores the effects of the net on dividuals, organizations, and societal :uctures. "Contributions Toward :veloping a Theoretical Framework" ntains two chapters. The first compares e printing press, USENET, and the ternet. At the time of its invention, the inting press created both communica-)n and information revolutions. This rt of the book presents USENET's and e Internet's potential for creating other, much grander, communication d information revolution.

this day of ubiquitous modems, the ternet, Internet Providers, and personal mputers, one sometimes forget there s a time when these things were not sely available. Many people and organi- ions were responsible for the creation the Internet and USENET. Much ınks should to go to the Department of fense for funding the early research. ıong the many people involved, several od out. J.C.R. Licklider and Robert /lor are two names associated with the ınding of the Internet. They saw the nputer as a communications tool with bal connectivity and as a way to share :h computer and human resources. is perspective was a very radical idea in 58, when computers from different nufacturers could not exchange data :ommunicate with each other. With partment of Defense research dollars l the Advance Research Projects Agency (ARPA), Licklider solved the immediate problems of getting incompatible computers to talk. But he never lost his global vision. His efforts resulted in the computer communications networks (ARPAnet). The global Internet can trace its roots back to this simple ARPAnet.

What if you were a poor, underendowed university without Defense Department research dollars? How could you get your computers talking to each other? Enter the "poor man's ARPAnet." Tom Truscott, Jim Ellis, and Steve Bellovin all had a desire to automatically share files and articles among several computer platforms. Fortunately, they were university students and cash poor. So they did the only thing they could do: they acquired some university computer time and an auto dialer, and applied a little creative UNIX hacking (the positive kind). Using these limited resources, these fellows developed what is now known as USENET. Their first incarnation of USENET simply dialed another computer, checked for new files, and then copied all the new files to itself. They set up their first USENET network on three university computers. Within a few years, these three nodes grew into several hundred nodes and eventually became part of the Internet.

This book illustrates that ordinary people with limited resources and a vision can make a difference. The grassroot's creation of USENET by Tom Truscott, Jim Ellis, Steve Bellovin, and others is such an example. People with significant resources and a vision can solve a specific, localized problem and simultaneously lay the foundation for solving global needs. Licklider's refusal to set his sights lower than the vision of a global computer is an example of exceeding short-term expectations. Look closely while reading the book, and you may find yourself viewing the world a little bit differently when you finish.

Bob Glickstein

Writing GNU Emacs Extensions

O'Reilly & Associates, 1997. ISBN 1-56592-261-1. Pp. 215. $ 29.95.

Reviewed by Eric Chin

<ericc@tera.com>

This is a good book. However, I would not purchase it for use as a reference manual. It does contain an appendix titled "Lisp Quick Reference," which valiantly enumerates the essentials of elisp. It does have a usable index, although it is not as outstanding as that of Wright and Steven's *TCP/IP Illustrated*, vol. 2. Those two features notwithstanding, this book doesn't purport to be a substitute for the Free Software Foundation's *GNU Emacs Lisp Reference Manual.*

Instead, this book is a great self-study text for a GNU Emacs Lisp 101 class. Coupled with lots of free time, read access to the elisp files that accompany a GNU Emacs distribution, and interactive access to a GNU Emacs editor, one can steadily work through this book chapter by chapter, example by example, and amass a personalized collection of elisp functions.

In my case, each page frequently yielded a new insight or trick. Within two weeks, my review copy had become well marked, with portions of every page I had read highlighted in yellow. Even the footnotes contain useful trivia. For example, did you know that the default keybinding for "eval-expression" recently changed ? Depending on the host I log on to, I either use xemacs version 19.10 or GNU Emacs 19.34, and I was puzzled by the latter version's unwillingness to react to "ESC-ESC." This was a minor enough issue that I did not pursue it, but simply resorted to using "M-x eval-e TAB" instead. However, when I finally got around to reading page 9 of this book, a footnote there immediately shed light on the matter, explaining that with the

advent of GNU Emacs 19.29, the default keybinding had been changed from "M-ESC" to "M-:".

As with any programming text, this book is best read within one arm's length of an interactive terminal, preferably one rendering a GNU Emacs window. Then, when confronted with an unfamiliar built-in elisp function, one needs only to use "M-x describe-func TAB" to retrieve the built-in documentation for that function. Even better, as one encounters each elisp example, one can immediately try it out by having Emacs evaluate it.

As a programmer, I often insert the current date into comments embedded within source code I edit. For this, I resort to "C-u M-! date," coupled with a little manual editing. However, for those who prefer elegance, chapter 4 presents a simple function that accomplishes this simple task, then evolves it into a set of functions that modify all time stamps within an Emacs buffer.

Therein lies the approach taken by this book toward the goal embodied in its title. The capabilities and secrets of elisp are presented as a side effect of accomplishing various editing tasks. Each function builds on and references concepts and constructs introduced in earlier sections. This is an effective presentation, but it obligates the reader to work through the book sequentially. One would be hard-pressed to randomly open the book at, say, chapter 7 and make much sense of it, without first having digested chapters 2, 3, 4, 5, and 6.

Nevertheless, if you are an Emacs enthusiast, I would recommend this book: not for your bedside bookshelf, but for that stack of books on your desk that you read while waiting for the compiler to rebuild an entire program suite.

James E. McDermott and John E. Phillips

Administering Usenet News Servers

Addison Wesley Developers Press, 1997. ISBN: 0-201-41967-X. Pp. 384. $39.95.

Reviewed by Nick Christenson

<npc@jetcafe.org>

It has always seemed strange to me that with all the books available on every conceivable aspect of Internet service, Usenet News has been relatively ignored by book publishers. There are a few books on using Usenet News and the venerable *Managing UUCP and Usenet* (O'Reilly & Associates, 1991), but there's nothing available on managing today's Usenet News services. This book attempts to fill that gap.

The authors start out with an overview of the Internet, which is pretty stock stuff, and the history of Usenet News, which isn't too bad, although the overview in Peter Salus's *Casting the Net* is better. Then they move to a chapter that covers the basics: what is news, a news server, a news feed, etc. It's a standard and straightforward introduction to the general topic of Usenet News.

The second section, "Planning," goes through a large number of steps that a prospective News admin should consider before installing the software. This section is a good idea, and there are some useful ideas in there, but the authors' recommendations on disk storage and bandwidth requirements for a full feed would have been woefully inadequate for Usenet News service even a year before publication of the book. The requirements for sizing the hardware of the server don't take into account the relative number of concurrent reader processes or number of feeds, and some aren't up to the task of handling a full feed in any case. The data provided in Brian Wong's *Configuration and Capacity Planning for Solaris Servers* (Prentice Hall, 1997) is much better, although still insufficient.

The authors do a fairly good job of evaluating the relative merits of INN and DNEWS, and they adequately cover security and network topology considerations. Their emphasis on PERT charts as part of the service construction process is misplaced, but they do a decent job of summarizing resources available on the Internet.

The third section, "Construction," deals with the building of the News server. There are several chapters, like "Installing the Operating System" and "Configuring the Operating System" that don't belong in a book like this unless they are specifically geared toward tuning or software requirements that apply specifically to News servers. Since this book's coverage of these issues is generic, I wish there had been references to other resources rather than chapters on these topics.

Finally, the fourth section, on administration, covers policies and maintenance. One place where the book excels is in discussing the corporate policy issues of running News service. This is usually overlooked by the site service implementors, and it is good to see this topic given some serious coverage.

On the other hand, the maintenance chapter contains nothing but a more detailed description of the major control files for INN and DNEWS. I don't know as much about DNEWS, but this information is pretty well covered for INN in the Installation Guide and manual pages which come with the distribution. While the book's treatment is a little clearer, it is not strictly necessary. There are also four appendices, of which Appendix D, NNTP Information with its list of NNTP commands and NNTP response codes, is the most interesting.

The book includes a CD which contains a recent version of INN, 1.5.1, an evaluation copy of DNEWS, and some miscellaneous utilities. Frankly, for a prospective news administrator, finding these distributions on the Internet, given informati

ın their locations in the text, is the least
f their difficulties; and the book would
ave been just as useful and probably $10
heaper without the CD.

inally, there are several errors, both syn-
actical and typographical, throughout
ıe text that are distracting at best and
ıisleading at worst. Some will grate on
ıe nerves of an experienced News
dministrator. Using a capital "H" for the
ktension on C language header files,
eferring to multiple spam messages as
spams," and many other distractions
ıouldn't appear in a serious technical
xt.

ın the whole, *Administering Usenet News*
ervers does a passable, if unspectacular,
b of getting someone who doesn't
ıow anything about Usenet to the brink
setting up a News server, but there's
ɔthing here at all for the News adminis-
ator with even a passing familiarity
ith the software. Recommended only if
ɔu need to set up a News server from
ratch and don't know where to start.

lward Yourdon

---

ath March

ntice Hall, 1997. ISBN: 0-13-748310-4. Pp. 218.
4.95.

viewed by Nick Christenson

pc@jetcafe.org>

s rare that I get excited about a com-
ıter book that has no hexadecimal
ımbers in it, but this is a good one.
), it's excellent. The subtitle is "The
ɔmplete Software Developer's Guide
Surviving Managing "Mission
ıpossible' Projects." Almost everyone in
computer industry has noticed their
ɔrkload increasing, while at the same
ıe the deadlines imposed by their
ɔervisors are shortening. Demands for
alified individuals, the amount of
ɔney at stake in the computer industry,
l the constraints of working on
ıternet time," have led to a showdown of corporate bottom line versus personal sanity. This book is your guide on how to deal with it.

Yourdon starts out describing what a Death March project is and explaining why they happen and why people participate in them. The book then goes on to cover such significant topics as how a Death March project team leader should negotiate with management for budget, working conditions, staffing, etc.; how to handle the day-to-day issues of working on such a project, and, just as importantly, when to know when it's time to get out.

I found this book to be remarkably refreshing in dealing with the true issues of working in these sorts of environments, and it is far more practical than I had expected it would be. It would have been easy (and entirely expected) for an author to say, "It's tough using a new formal process approach under such circumstances, so try not to." But Yourdon goes on to explain how to decide if a new system is appropriate for a Death March project and how to negotiate trying to get out of using it when it's not. This remarkably utilitarian approach pervades the whole book. It is clearly written by someone who has "been there" and has no use or time for trite answers. Talking about what to do when negotiating working conditions necessary for successful completion of the project fails, Yourdon says, "It's important to realize here that I'm not recommending resignation as a form of punishment or revenge. It's simply the rational thing to do when faced sith an impossible situation, ..." This sort of personal honesty is missing from the majority of books on computer careers.

The book focuses heavily on Death March projects as programming projects, despite the fact that the issues involved can apply equally to other computer projects or even completely different fields. For example, it's not much of a stretch to consider a team of accountants with one week to clean up the books completely before an audit to be on a Death March project. It would have been interesting to see some of the examples in the book extended along these lines.

This book was written with a great deal of collaboration from Internet denizens, who regaled Yourdon with tales of woe and success (mostly woe) from their own experiences. While this has led to a depth of coverage I wouldn't have expected for a first sojourn into the topic, a few things that weren't considered.

For example, Yourdon lists many reasons why someone may agree to sign up for a Death March project, but one notably absent is that completion of a project, such as building a new set of class libraries or building a new distributed data system, may lead to reduced work ahead. Also, there is probably a great deal of material to be written on the concept of having to complete continuous small projects over a very large span of time, as one might in, say, a Web page design outfit. These pose different but similar stresses and strains on the human psyche. It will be interesting to see what new issues will be addressed in the almost certain second edition.

On the whole, this book is excellent at covering its topic. It is certain to occasion thousands of thoughtful debates all over the world. At the very least, it will provide a great deal of practical advice for those "on the march" right now, as well as at least some measure of relief by letting the participants know that they're not alone. Expect many thousands of copies to make their ways to personal bookshelves all over Silicon Valley and the rest of the Internet world.

7th USENIX Security Symposium
January 26-29, 1998 • San Antonio, Texas

Opening Remarks
Avi Rubin, *AT&T Labs – Research*

Keynote Address: Security Lessons From All Over
Bill Cheswick, *Lucent Technologies, Bell Labs*

Architecture

A Comparison of Methods for Implementing Adaptive Security Policies
Brian Loe and Michael Carney, *Secure Computing Corporation*

The CRISIS Wide Area Security Architecture
Eshwar Belani, Amin Vahdat, Thomas E. Anderson, Michael Dahlin, *University of California at Berkeley*

Intrusion Detection

Bro: A System for Detecting Network Intruders in Real-Time
Vern Paxson, *Lawrence Berkeley National Laboratory*

Cryptographic Support for Secure Logs on Untrusted Machines
Bruce Schneier and John Kelsey, *Counterpane Systems*

StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks
Crispan Cowan, *Oregon Graduate Institute*

Data Mining Approaches for Intrusion Detection
Wenke Lee and Salvatore J. Stolfo, *Columbia University*

Network Security

Securing Classical IP Over ATM Networks
Carsten Benecke and Uwe Ellermann, *Universitaet Hamburg, Fachbereich Informatik*

A Java Beans Component Architecture for Cryptographic Protocols
Pekka Nikander and Arto Karila, *Helsinki University of Technology*

Secure Videoconferencing
Peter Honeyman, Andy Adamson, Kevin Coffman, Janani Janakiraman, Rob Jerdonek, and Jim Rees, *CITI, University of Michigan*

Distributed Systems

Unified Support for Heterogeneous Security Policies in Distributed Systems
Victoria Ungureanu and Naftaly H. Minsky, *Rutgers University*

Operating System Protection for Fine-Grained Programs
Trent Jaeger, Jochen Liedtke, and Nayeem Islam, *IBM T.J. Watson Research Center*

Expanding and Extending the Security Features of Java
Karen R. Sollins and Nimisha V. Mehta, *MIT Laboratory for Computer Science*

World Wide Web Security

Towards Web Security Using PLASMA
A. Krannig, *Fraunhofer-Institute for Computer Graphics IGD*

Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies
Vinod Anupam and Alain Mayer, *Bell Labs, Lucent Technologies, Bell Labs*

Finite-State Analysis of SSL 3.0
John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern, *Stanford University*

Cryptography

Certificate Revocation and Certificate Update
Kobbi Nissim and Moni Naor, *Weizmann Institute of Science*

Attack-Resistant Trust Metrics for Public Key Certification
Raph Levien and Alex Aiken, *University of California at Berkeley*

Software Generation of Random Numbers for Cryptographic Purposes
Peter Gutmann, *University of Auckland*

Invited Talk: The Security Product Market: Trends and Influences
Marcus Ranum, *Network Flight Recorder, Inc.*

Panel: Computer Security and Legal Liability
Moderator: Steve Bellovin, *AT&T Labs – Research*

Panelists: Ed Cavazos, *Interliant Corporation*
Others to be announced.

Invited Talk: Factoring: Facts and Fables
Arjen K. Lenstra, *Citibank, N.A.*

Invited Talk: Elliptic Curves—Ready for Prime Time
Alfred Menezes, *Auburn University*

Invited Talk: Securing Electronic Commerce: Applied Computer Security or Just Common Sense
Clifford Neuman, *University of Southern California*

Invited Talk: Real World Security Practices
JoAnn Perry, *Independent Consultant,* and Shabbir Safdar, *Goldman, Sachs & Co.*

Work-In-Progress Reports (WIPs)

The Works-In-Progress session will consist of five minute presentations. Speakers should submit a one or two paragraph abstract to *sec98wips@usenix.org* by January 15. Please include your name, affiliation, and the title of your talk. Please note this is a change from the original instructions in the Call for Papers. A schedule of presentations will be posted at the conference by Noon on January 29. Experience at other conferences has shown that most submissions are usually accepted. The five minute time limit will be strictly enforced.

Announcement and Call for Participation

3rd USENIX Workshop on Electronic Commerce

August 31–September 3, 1998
Tremont Hotel
Boston, MA

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates

Extended abstracts due: *March 6, 1998*
Notification to authors: *April 17, 1998*
Camera-ready final papers due: *July 21, 1998*

Program Committee

Chair: Bennet S. Yee, *UC San Diego*
Ross Anderson, *Cambridge University*
Nathaniel Borenstein, *First Virtual*
Marc Donner, *Morgan Stanley*
Niels Ferguson, *Digicash*
Mark Manasse, *Digital Equipment Corp.*
Cliff Neuman, *University of Southern California*
Avi Rubin, *AT&T Labs*
Win Treese, *OpenMarket*
Doug Tygar, *Carnegie Mellon University*
Hal Varian, *UC Berkeley*

Overview

The Third Workshop on Electronic Commerce will provide a major opportunity for researchers, experimenters, and practitioners in this rapidly self-defining field to exchange ideas and present the results of their work. It will set the technical agenda for work in electronic commerce by enabling workers to examine urgent questions, share their insights and discover connections with other work that might otherwise go unnoticed. To facilitate this, the conference will not be limited to technical problems and solutions, but will also consider their context: the economic and regulatory forces that influence the engineering choices we make, and the social and economic impact of network based trading systems.

Tutorials Proposals Welcome

One day of tutorials will precede the Workshop on August 31. USENIX's well-respected tutorials are intensive and provide immediately-useful information delivered by skilled instructors who are hands-on experts in their topic areas. Topics for the Electronic Commerce Workshop will include, but are not limited to, security and cryptography. If you are interested in presenting a tutorial, please contact:

Dan Klein, Coordinator
Email: *dvk@usenix.org*
Phone: 412.421.2332

Workshop Topics

Two and one-half days of technical sessions will follow the tutorials. We welcome submissions for technical and position paper presentations, reports of work-in-progress, technology debates, and identification of new open problems. Birds-of-a-Feather sessions in the evenings and a keynote speaker will round out the program.

We seek papers that address a wide range of issues and ongoing developments, including, but not limited to:

- Advertising
- Anonymous transactions
- Auditability
- Business issues
- Copy protection
- Credit/Debit/Cash models
- Cryptographic security
- Customer service
- Digital money
- EDI
- Electronic libraries
- Electronic wallets
- Email-enabled business
- Exception handling
- Identity verification
- Internet direct marketing
- Internet/WWW integration
- Key management
- Legal and policy issues
- Micro-transactions
- Negotiations
- Privacy
- Proposed systems
- Protocols
- Reliability
- Reports on existing systems
- Rights management
- Service guarantees
- Services vs. digital goods
- Settlement
- Smart-cards

Questions regarding a topic's relevance to the workshop may be addressed to the program chair via electronic mail to *ec98chair@usenix.org*. USENIX will publish Conference Proceedings which are provided free to technical session attendees; additional copies will be available for purchase from USENIX.

What to Submit

Technical paper submissions and proposals for panels must be received by March 6, 1998. We welcome submissions of the following type:

1. Refereed Papers—Full papers or extended abstracts should be five to 20 pages, not counting references and figures.
2. Panel proposals—Proposals should be three to seven pages, together with a list of names of potential panelists. If accepted, the proposer must secure the participation of panelists, and prepare a three to seven page summary of panel issues for inclusion

in the Proceedings. This summary can include position statements by panel participants.

3. Work-In-Progress Reports—Short, pithy, and fun, WIP reports introduce interesting new or ongoing work and should be 1 to 3 pages in length. If you have work you would like to share or a cool idea that is not quite ready to publish, a WIP is for you! We are particularly interested in presenting student work.

Each submission must include a cover letter stating the paper title and authors, along with the name of the person who will act as the contact to the program committee. Please include a surface mail address, daytime and evening phone number, email and fax numbers and, if available, a URL for each author. If all of the authors are students, please indicate that in the cover letter for award consideration (see "Awards" below).

USENIX workshops, like most conferences and journals, require that papers not be submitted simultaneously to more than one conference or publication and that submitted papers not be previously or subsequently published elsewhere. Submissions accompanied by non-disclosure agreement forms are not acceptable and will be returned to the author(s) unread. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976.

Where to Submit Proposals

Please send submissions to the program committee via one of the following methods. All submissions will be acknowledged.

- Preferred Method: email (Postscript or PDF formats only) to: *ec98papers@usenix.org.*
- Files should be encoded for transport with uuencode or MIME base64 encoding.

Authors should ensure that the PostScript is generic and portable so that their papers will print on a broad range of postscript printers, and should submit in sufficient time to allow us to contact the author about alternative delivery mechanisms in the event of network or other failure. If you send PostScript, remember the following:

- Use only the most basic fonts (TimesRoman, Helvetica, Courier). Other fonts are not available with every printer or previewer.
- PostScript that requires some special prolog to be loaded into the printer won't work for us. Please don't send it.
- If you use a PC- or Macintosh-based word processor to generate your PostScript, print it on a generic PostScript printer before sending it, to make absolutely sure that the PostScript is portable.
- If you are generating the PostScript from a program running under Windows, make sure that you establish the "portable" setting, not the "speed" setting for PostScript generation.

  A good heuristic is to make sure that recent versions of Ghostview (e.g. Ghostview 1.5 using Ghostscript 3.33) can display your paper.

- **Alternate Method:** 10 copies, via postal delivery to:

  EC'98 Submissions
  USENIX Association
  2560 Ninth Street, Suite 215
  Berkeley, CA 94710

For detailed submission guidelines, send email to *ec98authors@usenix.org,* refer to the conference Web page at *www.usenix.org/events/ec98/guidelines.html,* or send email to the program chair at *ec98chair@usenix.org.*

An electronic version of this Call for Papers is available at: *www.usenix.org/events/ec98/.*

Birds-Of-A-Feather Sessions (BoFs)

Do you have a topic that you'd like to discuss with others? Our Birds-of-a-Feather Sessions may be perfect for you. BoFs are very interactive and informal gatherings for attendees interested in a particular topic. Schedule your BoF in advance by telephoning the USENIX Conference Office at 714.588.8649 or sending email to: *conference@usenix.org.*

Awards

The program committee will offer awards of $500 for the best paper and the best student paper.

Registration Information

Materials containing all details of the technical and tutorial programs, registration fees and forms and hotel information will be available in June, 1998. If you wish to receive the registration materials, please contact USENIX at:

USENIX Conference Office
22672 Lambert Street, Suite 613
Lake Forest, CA 92630
Phone: 714 588 8649
Fax: 714 588 9706
Email: *conference@usenix.org*

About USENIX

USENIX is the Advanced Computing Systems Association. Since 1975 USENIX has brought together the community of engineers, system administrators, and technicians working on the cutting edge of the computing world. For more informaiton about USENIX:

URL: *www.usenix.org*
Email: *office@usenix.org*
Fax: 510.548.5738
Phone: 510.528.8649

Announcement and Call for Participation USENIX

Tcl/Tk '98: The 6th Annual Tcl/Tk Conference

September 14–18, 1998
San Diego, California

Sponsored by USENIX, the Advanced Computing Systems Association

Important Dates:

Paper, Demonstrations, and Panel Proposals: *April 8, 1998*
Acceptance notification: *May 11, 1998*
Poster submissions: *June 26, 1998*
Camera-ready copy: *July 28, 1998*

Conference Organizers:

Co-chairs:
Don Libes, *NIST*
Michael McLennan, *Bell Labs Innovations for Lucent Technologies*

Program Committee:
Dave Beazley, *University of Utah*
De Clarke, *UCO/LICK Observatory*
Dave Griffin, *Digital Equipment Corporation*
Mark Harrison, *AsiaInfo Holdings, Inc.*
Jeffrey Hobbs, *Siemens AG*
George Howlett, *Bell Labs Innovations for Lucent Technologies*
Ray Johnson, *Sun Microsystems Laboratories, Inc.*
Kevin Kenny, *General Electric Corporate R&D*
Tom Phelps, *University of California at Berkeley*
Tom Poindexter, *Talus Technologies, Inc.*
John Reekie, *University of California at Berkeley*
Forest Rouse, *ICEM CFD Engineering*
Alex Safonov, *University of Minnesota*
Henry Spencer, *SP Systems*

The Sixth Annual Tcl/Tk Conference is a forum to:

- bring together Tcl/Tk researchers and practitioners
- publish and present current work involving Tcl/Tk
- learn about the latest developments in Tcl/Tk
- plan for future Tcl/Tk related developments

The conference program will include formal paper and panel presentations, poster and demonstration sessions, works in progress (WIP) sessions, Birds of a Feather (BOF) sessions, and tutorials.

Overview and Forms of Participation

All forms of participation provide an opportunity to report on original Tcl/Tk research. The audience is practitioners and researchers who are experienced users of Tcl/Tk. For this reason, reports on experiences and applications must draw out lessons for other Tcl/Tk developers. Topics include, but are not limited to:

- System extensions
- Novel Tcl/Tk-based applications
- Experience reports on building applications in Tcl/Tk
- Comparative evaluations of Tcl/Tk and other languages or toolkits for building applications
- Use of different programming paradigms in Tcl/Tk and proposals for new directions.

Papers

Papers should be concise. Omit extraneous or redundant text. Length is not a direct factor in judging paper quality; however, historically, most papers ar e 12 pages or less. Consider trimming longer papers, possibly by splitting into separate and more focused papers. Authors of accepted papers will have twenty minutes to present the paper at the conference. Full papers written in English must be submitted for review. Authors are encouraged to include black-and-white figures in their papers.

The program committee will review and evaluate papers according to the following criteria:

- Quantity and quality of novel content
- Relevance and interest to the Tcl/Tk community
- Quality of presentation of content in the paper
- Suitability of content for presentation at the conference

Papers should be 8–12 single-spaced pages and should present a cohesive piece of work. Papers so short as to be considered extended abstracts will not receive full consideration. Papers may report on commercial or non-commercial systems, but those with blatant marketing content will not be accepted.

Application and experience papers need to strike a balance between background on the application domain and the relevance of Tcl/Tk to the application. Application and experience papers should clearly explain how the application or experience illustrates a novel use of Tcl/Tk, and what lessons the Tcl/Tk community can derive from the application or experience to apply to their own development efforts.

This conference requires that papers not be submitted simultaneously to other conferences or publications, and that

submitted papers not be previously published or accepted papers subsequently published elsewhere for a period of one year after acceptance. Papers accompanied by non-disclosure agreement forms will be returned to the author(s) unread. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976.

Posters

Poster sumissions provide an opportunity to present interesting or preliminary results. They are the ideal category for material that is better suited for discussion in small groups as opposed to large groups.

Posters will be displayed during one day of the conference. A poster session will provide an opportunity for attendees to interact with poster authors individually and in small groups. Display space will be approximately 3 feet wide by 4 feet high. Poster authors should submit a draft of their poster's contents along with a one-page abstract. Abstracts of accepted posters will be published in the conference proceedings.

Demonstrations

There will be a demonstration reception one evening. Demonstrations will be held in parallel, allowing attendees to more closely interact with the demonstrators. Space will be available for demonstrations in the following categories:

Reviewed demonstrations

- will be given a demonstration station for the entire session and
- will have an abstract published in the conference proceedings.

Submissions should include both a one-page abstract and six copies of a videotape (VHS) showing the demonstration. Some demonstrations may also be scheduled for a conference session.

Informal demonstrations

- will be assigned a specific time during the demonstration session.

Authors of accepted papers as well as those with demonstration-ready Works-in-Progress are encouraged to sign up for informal demonstration time slots. More information on the facilities available for informal demonstrations will be provided in the registration materials and on the conference Web site.

Demonstrations of commercial products of interest to the Tcl/Tk community are encouraged. The abstract for the proceedings, however, should avoid commercial content (i.e., it should not include pricing and sales information or marketing content).

Panel Proposals

The program committee is organizing panel discussions of up to 90 minutes. Proposals should include a list of confirmed panelists, a title and format, and a panel description with position statements from each panelist. Panels should have no more than four speakers, including the panel moderator, and should allow time for substantial interaction with attendees. Panels are not presentations of related research papers. Papers should be submitted individually and the program committee will group them into sessions of related material.

Works-in-Progress Presentations and Birds-of-a-Feather Sessions

Works-in-Progress (WIP) presentations and Birds-of-a-Feather sessions (BOFs) are not reviewed. Slots for both are available on a first-come, first-served basis starting in August 1998. Specific instructions for reserving WIP and BOF time slots will be provided in the registration information in June 1998. Some WIP and BOF time slots will be held open for on-site reservation, so we encourage all attendees with interesting work in progress to consider presenting that work at the conference.

Tutorials

On Monday and Tuesday, September 14–15, USENIX's well-respected tutorial program will offer intensive, immediately useful, half- and full-day sessions. Skilled instructors who are hands-on experts in their topic areas present both introductory and advanced tutorials.

How to Submit a Paper, Demonstration, or Panel Proposal

We are accepting most conference submissions electronically, via email. Paper, poster, panel and demonstration proposal submissions should be sent in two forms:

- Postscript or PDF files formatted for an 8.5 × 11 inch page with 1 inch margins. (Be sure that it will print on a variety of printers.)
- Plain text or HTML (standard markups only, no browser-specific tags).

Send submissions to:
tcl98papers@usenix.org

If accepted, both electronic and camera-ready hardcopy of the final version (full paper, poster abstract, or panel summary and position statements) will be required.

Detailed submission instructions are available at the conference Web site: *http://www.usenix.org/events/tcl98/*, or by sending email to *tcl98authors@usenix.org*.

Demonstrations should also submit six copies of a VHS videotape showing the demonstration. The videotapes are for review purposes only, and cannot be returned. If accepted, both camera-ready and electronic versions of the abstract will be required.

Postal Address:
Tcl/Tk 98 Conference
USENIX Association
2560 Ninth Street, Suite 213
Berkeley CA 94710
Phone: 510.528.8649

More information on the conference will be available at the conference Web site: *http://www.usenix.org/events/tcl98* .

Registration Materials

Materials containing all details of the technical and tutorial programs, registration fees and forms, and hotel information will be available in June, 1998. If you wish to receive the registration materials, please contact:

USENIX Conference Office
22672 Lambert Street, Suite 613
Lake Forest CA 92630
714.588.8649
Fax: 714.588.9706
Email: *conference@usenix.org*

Announcement and Call for Papers

3rd Symposium on Operating Systems Design and Implementation (OSDI '99)

February 22–Feb 25, 1999
New Orleans, LA, USA

Sponsored by USENIX, the Advanced Computing Systems Association
Co-sponsored by IEEE TCOS and ACM SIGOPS (pending)

Important Dates

Full papers due: *July 28, 1998*
Notification to authors: *October 13, 1998*
Revised papers due for shepherding: *December 1, 1998*
Camera-ready full papers due: *January 6, 1999*

Program Committee

Co-Chairs
Margo Seltzer, *Harvard University*
Paul Leach, *Microsoft*

Tom Anderson, *University of Washington*
John Hartman, *University of Arizona*
Kai Li, *Princeton University*
Bruce Lindsay, *IBM Almaden Research Center*
Nancy Lynch, *Massachusetts Institute of Technology*
Greg Minshall, *Ipsilon*
Sape Mullender, *University of Twente*
Michael O'Dell, *UUNET Technologies*
Sean O'Malley, *Network Appliance*
Rob Pike, *Lucent Technologies*

Continuing in the tradition of the OSDI symposia, the third OSDI will continue to focus on practical issues related to modern operating systems. OSDI brings together professionals from academic and industrial backgrounds and has become the perfect forum for issues concerning the design and implementation of operating systems for modern computing platforms such as workstations, parallel architectures, mobile computers, and high speed networks.

The OSDI symposium emphasizes both innovative research and quantified experience in operating systems. We seek contributions from all fields of operating systems practice: new ideas, the influence of new hardware and networking on systems, evaluations of existing systems, etc. The stress is on practice: What can we learn from trying to build systems that work well? Sometimes they don't work well; OSDI encourages the submission of papers that discuss the reasons for systems' failures as well as successes.

Submission Process

Authors are required to submit full papers by July 28, 1998. Submitted papers should be no longer than 14 pages, spaced no closer than standard 10 point font on 12 point baseline, single- or double-column format. Longer submissions will be discarded without review. Very similar papers must not have been published or submitted for publication elsewhere. All submissions will be held in the highest confidentiality prior to publication. Papers accompanied by non-disclosure agreement forms are not acceptable and will be returned unread.

The papers will be judged on interest, significance, originality, clarity, relevance, and correctness. The committee will favor papers with reproducible results, especially those supplying detailed data and explanations, or offering to make data sets or source code available. Accepted papers will be shepherded through an editorial review process by a member of the program committee.

Authors of accepted papers must provide an HTML page containing the abstract and links to their paper, slides, and software, if available. This will be collected after the event for inclusion in an electronic version of the symposium. For examples, see
www.cs.utah.edu/~lepreau/osdi94/ and
www.usenix.org/publications/library/proceedings/osdi96/

Best Paper Awards

Awards will be given for the best paper at the conference and best student paper.

Where to Submit

Submission of all papers must be made in both paper and electronic form. Fifteen (15) paper copies (double-sided if possible) of the paper must be sent to:

Margo Seltzer
Division of Engineering and Applied Science
Harvard University
28 Oxford St.
Cambridge, MA 02138
617.496.5663

and one electronic copy in PostScript (not ASCII) must be submitted by electronic mail to *osdi99papers@usenix.org*

For administrative reasons (not blind reviewing), every submission (in both its paper and electronic form) should include one additional page containing (i) paper title and authors, indicating any who are full-time students, and (ii) for the author who will act as the contact to the program committee, his or her name, paper mail address, daytime and evening phone numbers, email address and fax number, if available. The cover sheet mailed with the electronic paper submission should be in ASCII to facilitate accurate on-line bookkeeping and should be included in the same electronic mail message as the PostScript file containing the paper.

For more details on the submission process, authors are encouraged to consult

www.usenix.org/events/osdi99/guidelines.html

or send mail to *osdi99authors@usenix.org*.

All submissions will be acknowledged by September 8, 1998. If your submission is not acknowledged by this date, please contact the program chairs promptly at *osdi99chairs@usenix.org*.

Symposium Overview

The symposium will consist of one day of tutorials, followed by two and one-half days of single-track technical sessions with presentations of the refereed papers, and a half-day workshop on a topic to be determined. One of the technical sessions will be dedicated to work-in-progress presentations (WIPS) and will be described in later announcements. The refereed papers will be published in the Proceedings, provided free to technical session attendees and available for purchase from USENIX. The Proceedings may also be distributed to ACM SIGOPS members.

Registration Materials

Materials containing all details of the technical and tutorial programs, registration fees and forms, and hotel information will be available in November, 1998. If you wish to receive the registration materials, please contact:

USENIX Conference Office
22672 Lambert Street, Suite 613
Lake Forest, CA 92630
714.588.8649
Fax: 714.588.9706
Email: *conference@usenix.org*
URL: *www.usenix.org*

A UNIQUE OFFER ON THE BEST IN UNIX FOR USENIX MEMBERS

20% DISCOUNT FROM McGRAW-HILL

☐ **THE INTERNET GUIDE FOR NEW USERS**
D. Dern
hardcover, 016510-6, $40.00,
MEMBER PRICE **$32.00**
paperback, 016511-4, $27.95,
MEMBER PRICE **$22.36**

☐ **INTERNET FOR EVERYONE**
R. Wiggins
hardcover, 067018-8, $29.95,
MEMBER PRICE **$23.96**
paperback, 067019-6, $45.00,
MEMBER PRICE **$36.00**

☐ **THE ESSENTIAL INTERNET INFORMATION GUIDE**
J. Manger
707905-1, paperback, $27.95,
MEMBER PRICE **$22.36**

☐ **THE INFORMATION BROKERS HANDBOOK, Second Edition**
S. Rugge
911878-x, paperback, $34.95,
MEMBER PRICE **$27.96**
Available December 1994

☐ **SAA AND UNIX: IBM's Open System Strategy**
M. Killen
034607-0, $40.00,
MEMBER PRICE **$32.00**

☐ **A STUDENT'S GUIDE TO UNIX**
H. Hahn
025511-3, paperback, $28.00,
MEMBER PRICE **$22.40**

☐ **UNIX DEVELOPER'S TOOL KIT**
K. Leininger
911836-4, $65.00,
MEMBER PRICE **$52.00**

☐ **UNIX SECURITY: A Practical Tutorial**
N. Arnold
002560-6, $24.95,
MEMBER PRICE **$19.96**

☐ **THE UNIX AUDIT: Using UNIX to Audit UNIX**
M. Grottola
025127-4, $32.95,
MEMBER PRICE **$26.36**

☐ **UNIX: A Database Approach**
S. Das
015745-6, $29.95,
MEMBER PRICE **$23.96**
Available November 1994

I am a member of USENIX Association. Please send me the books I have indicated at the member special rate. I have added $3.00 postage and handling for the first book ordered, $1.00 for each additional book, plus my local sales tax.

Check or money order is enclosed—payable to McGraw-Hill, Inc.
Charge my ❑ Visa ❑ Mastercard
❑ Discover ❑ Amex

Account #_____

Expiration Date_____

Signature_____

USENIX Membership # _____

Bill & Ship To:

Name_____

Street_____

City, State, Zip_____

Daytime Phone #_____

03US002

Send or Fax Orders to:

McGraw-Hill, Inc.
Attn: Rosa Perez
11 West 19th Street—4th Floor
New York, New York 10011
Fax: 212-337-4092

If I am not completely satisfied, I will return the book(s) within 15 days for a full refund or credit. Satisfaction unconditionally guaranteed. Prices subject to change without notics. We can only accept orders within the continental USA.

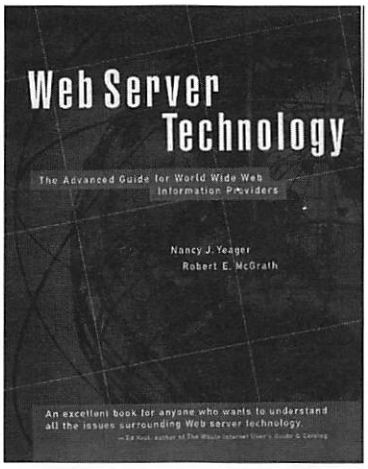
Web Server Technology

The Advanced Guide for World Wide Web Information Providers

Nancy J. Yeager and Robert E. McGrath

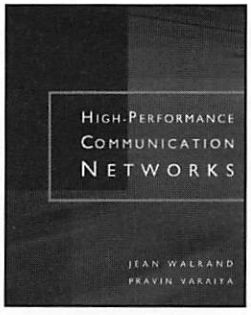
National Center for Supercomputing Applications

This authoritative presentation of web server technology takes you beyond the "how-to" guides to provide an understanding of the underlying principles and technical details of how Web servers really work. It explains the current state of the art and suggests how the technology will evolve. Topics covered include: performance, caching, search, security, and digital commerce.



1996
407 pages
Paperback
ISBN 1-55860-376-X
$34.95

"An excellent book for anyone who wants to understand all the issues surrounding Web server technology."

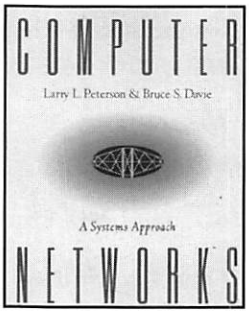


High-Performance Communication Networks

Jean Walrand and Pravin Varaiya, both of the University of California, Berkeley

Meet your evolving needs with this in-depth presentation of emerging technologies used to build high-speed, high-performance communication networks. It explains how the converging telephone, data, and CATV technologies are combined into high-performance networks, and how to plan, manage, and control these networks. The perspectives of the communications engineer, the computer scientist, and the economist are combined to provide a system-level understanding of the core networking principles and technologies.

"The experience of reading this book is almost as if I had spent a three day weekend seminar in a general discussion with an internetworking and engineering expert."



Computer Networks: A Systems Approach

Larry L. Peterson, University of Arizona and Bruce S. Davie, Cisco Systems, Inc.

A systems-oriented view of computer network design that goes beyond current technology to help you grasp the underlying concepts and build a foundation for making sound network design decisions. By providing an understanding of the components of a network and a feel for how these components fit together, this book empowers you to design real networks that are both efficient and elegant. It uses the Internet to illustrate how network protocols are designed.

"I was pleased with both the depth and clarity of the book. This is a winning combination. The discussions start with the basics, motivate...through examples and descriptions of the actual problems being solved, and end up with lively discussions of some of the current open problems."

Additional Titles of Interest

Introduction to Data Compression
Khalid Sayood, University of Nebraska, Lincoln
1996; hardcover; 475 pages; ISBN 1-55860-346-8;

Object-Relational DBMSs: The Next Great Wave
Michael Stonebraker, Informix-with Dorothy Moore
1996; paperback; 216 pages; ISBN 1-55860-397-2;

Intelligent Java Programming for the Internet and Intranets
Mark Watson, Angel Studios
1997; paperback; 500 pages; ISBN 1-55860-420-0;

MORGAN KAUFMANN PUBLISHERS

Phone: 800-745-7323 or 415-392-2665
Email: orders@mkp.com
http://www.mkp.com

Local User Groups

UNIX and LINUX Groups

The USENIX Association will support local user groups by doing a mailing to assist in the formation of a new group and publishing information on local groups in *:login;* and on its Web site. Full details can be found at: <http://www.usenix.org/membership/LUGS.html>.

At least one member of the group must be a current member of the Association.

Send additions and corrections to: <login@usenix.org>

California

Orange County

UNIX Users Association of Southern California (UUASC)

Colorado

Boulder

Boulder Linux Users Group

Front Range UNIX Users Group

Connecticut

The Connecticut Free UNIX Group

District of Columbia

Washington

Washington Area UNIX Users Group

Florida

Orlando

Central Florida UNIX Users Group

Western

Florida West Coast UNIX Users Group

Georgia

Atlanta

Atlanta UNIX Users Group

Kansas/Missouri

Kansas City

Kansas City UNIX Users Group (KCUUG)

Massachusetts

Worcester

WPI Linux Association

Worcester Linux User's Group

Michigan

Detroit/Ann Arbor

Southeastern Michigan Sun Local Users and Nameless UNIX Users Group

Missouri

St. Louis

St. Louis UNIX Users Group

New England

Northern New England UNIX Users Group (NNEUUG)

New Mexico

Albuquerque

ASIGUNIX

New York

New York City

Unigroup of New York City

Oklahoma

Tulsa

Tulsa UNIX Users Group, $USR

Texas

Austin

Capital Area Central Texas UNIX Users Society (CACTUS)

Dallas/Fort Worth

Dallas/Fort Worth UNIX Users Group

Houston

Houston UNIX Users Group (HOUNIX)

Washington

Seattle

Seattle UNIX Group

Armenia

Yerevan

The Armenian UNIX Users Group (AMUUG) was founded in December 1996. AMUUG is open to all interested individuals and organizations, regardless of affliation, without any fee.

Canada

Alberta

Calgary UNIX Users Society (CUUG)

Manitoba

Manitoba UNIX Users Group (MUUG)

Ontario

Toronto Group

Ottawa Carleton UNIX Users Group (OCUUG)

System Administration Groups

SAGE supports local groups. Full listing of group Web sites and other details can be found at:

http://www.usenix.org/sage/locals/

ASUQ
Meets first Wednesday of every third month in Montreal, Quebec.

AZSAGE
Meets monthly in the Phoenix area.

BayLISA
Serves the San Francisco Bay Area.

Back Bay LISA (BBLISA)
Serves the Boston, Massachusetts and New England area.

Beach LISA
A group for system administrators in the San Diego area hosts monthly meetings and the occasional social event.

dc.sage
Serves the Washington D.C. Area.

Dallas-Fort Worth SAGE (dfwsage)
Serves the North Texas area.

$GROUPNAME
Serves the New Jersey area.

EnglishBayLISA
Serves the Vancouver, British Columbia and the British Columbia lower mainland.

Houston Area Sysadmins (HASH)
Serves the greater Houston Area. Join the mailing list by sending a subscribe message to hash-request @tree.egr.uh.edu

Los Angeles Area Group
A group in Los Angeles is being formed. Please contact Josh Geller (joshua@cae.retix.com) if you are interested.

New York Systems Administrators (NYSA)
Serves the New York City area.

North Carolina System Administration Interest Group
Serving central North Carolina, particularly the Triangle Area.

Old Bay SAGE
A group for sysadmins in the greater Baltimore Maryland area.

SAGE-AU
The System Administrators Guild of Australia

Seattle SAGE Group (SSG)
Seattle, Washington area.

Twin Cities Systems Administrators (TCSA)
Serving the Twin Cities and surrounding areas of Minnesota.

motd

by Rob Kolstad

Rob Kolstad is president of BSDI and a long-time USENIX member, having served as chair of several conferences and workshops, director on the Board, and editor of *;login:*. He is also head coach of the USA programming team.

How 'bout that LISA conference? Over 2,000 people! It was pretty cool to see all those system administrators in one place.

I have this neat little card from Wilson Learning in Eden Prairie, Minnesota, a training company that instructs employees on various topics surrounding the workplace, including the notion of "working together." Great courses, by the way.

The particular course I took had to do with showing how people's "styles" can create apparent conflicts in the workplace, even when goals are shared and everyone is moving toward completion of a project. They explain many aspects of behavior and define various "social styles."

The card tells how to observe people and pigeonhole them into one of four boxes (which is, of course, not very many boxes). It then helps you remember how best to deal with them. Although individuals are a combination of millions of different traits and behaviors, it is possible to learn to better understand what they treasure, what "climate" makes them comfortable, and how to help them make decisions (among other things).

For instance people who are classified as "amiables" often value investments in relationships far more than the "drivers" (who really value time above all else). "Expressives," when backed into a corner, will attack. "Analyticals," on the other hand, will avoid the conflict.

It's a handy little card that helps me keep in mind that we are all individuals who see the world in different ways and attack problems from different angles. There are no value judgments, of course; all these various properties are just the ways of the world.

Why do I mention this? Because I find myself overwhelmed with deadlines today. Being a clever guy, the deadlines are all of my own making. I sometimes plan too optimistically! Regrettably, I am not so naive as to be unable to analyze my behavior in the face of this relatively tough set of deadlines.

I am sorry to report that, for the last hour or two, I have been an "avoider." I am spinning from task to task and the context switch overhead is, arguably, three times the time being devoted to each task. It is as if I am paging – or even swapping. I am trying to combat this by following a list of items that must be done in strict time order (phone calls to various time zones, FedEx deadline, that sort of thing). It sure is stressful compared to elementary school (when waiting for the recess bell was the height of my stress during those years). I hate not getting things in by deadlines, you know.

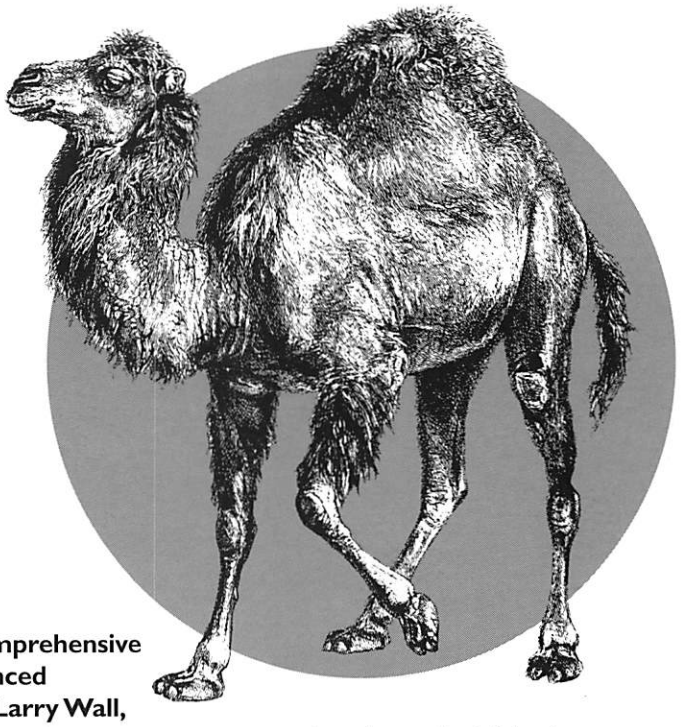
The little card would have me believe that my behavior is that of an "analytical," even when all the tests suggest that I am almost off the charts as a "driver." I sure hope I'm not mellowing out too much in my old age!

If you find yourself overwhelmed, you might try the notion of making a strict task list and following tasks through to completion (batch scheduling) in an effort to reduce the size of the list. It works for me on a good day.

UNIX EDITION

ANNOUNCING O'REILLY'S

Perl RESOURCE KIT

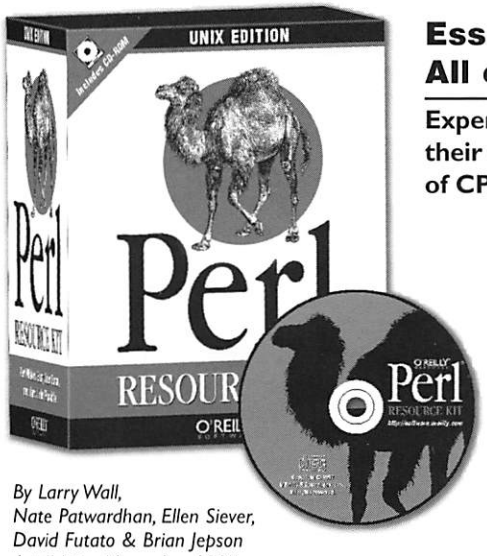


The *Perl Resource Kit—UNIX Edition* gives you the most comprehensive collection of Perl documentation and commercially enhanced software tools available today. Developed in association with Larry Wall, the creator of Perl, it's the definitive Perl distribution for webmasters, programmers, and system administrators.

The *Perl Resource Kit* provides:

- Over 1,800 pages of tutorial and in-depth reference documentation for Perl utilities and extensions, in 4 volumes.
- A CD-ROM containing the complete Perl distribution, plus hundreds of freeware Perl extensions and utilities—a complete snapshot of the Comprehensive Perl Archive Network (CPAN)—as well as new software written by Larry Wall just for the Kit.

10% Off
For USENIX Members Only



By Larry Wall, Nate Patwardhan, Ellen Siever, David Futato & Brian Jepson
1st Edition November 1997
1812 pages, ISBN 1-56592-370-7, $149.95
Includes 4 books & CD-ROM

Essential Perl Software Tools All on One Convenient CD-ROM

Experienced Perl hackers know when to create their own, and when they can find what they need on CPAN. Now all the power of CPAN—and more—is at your fingertips. The *Perl Resource Kit* includes:

- A complete snapshot of CPAN, with an install program for Solaris and Linux that ensures that all necessary modules are installed together. Also includes an easy-to-use search tool and a web-aware interface that allows you to get the latest version of each module.
- A new Java/Perl interface that allows programmers to write Java classes with Perl implementations. This new tool was written specially for the Kit by Larry Wall.

Experience the power of Perl modules in areas such as CGI, web spidering, database interfaces, managing mail and USENET news, user interfaces, security, graphics, math and statistics, and much more.

For more information, go to: *http://perl.oreilly.com/log*
or call 800-998-9938.

O'REILLY™ SOFTWARE

101 Morris Street, Sebastopol CA 95472 • For inquiries: 800-998-9938, 707-829-0515 • Weekdays 6AM-5PM PST
FAX: 707-829-0104 • Email a request for our catalog: *catalog@online.oreilly.com* • Check out our website:
http://software.oreilly.com/ • O'Reilly Technical Publications website: *http://www.oreilly.com/*

CONNECT WITH USENIX

MEMBERSHIP AND PUBLICATIONS

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 510 528 8649
FAX: 510 548 5738
Email: <office@usenix.org>

WEB SITE

http://www.usenix.org

AUTOMATIC INFORMATION SERVER

If you do not have access to the Web, finger <info@usenix.org> and you will be directed to the catalog which outlines all conferences, activities, and services.

PGP INFORMATION

All correspondence to:
Key ID: D6C05861 1996/11/11
USENIX 1997 <pgp@usenix.org>
6118D4DBE52612EB 8031C57DAE9AC168

Master key, for signatures only:
Key ID: 2FEA2EF1 1996/04/08

USENIX master key <not-for-mail>:
DBA 7509966E48AA9 80B2D9E2FEDA005E

USENIX

CONTRIBUTIONS SOLICITED

You are encouraged to contribute articles, book reviews, and announcements to *;login:*. Send them via email to <login@usenix.org> or through the postal system to the Association office.

Send SAGE material to <tmd@usenix.org>. The Association reserves the right to edit submitted material. Any reproduction of this magazine in its entirety or in part requires the permission of the Association and the author(s).

The closing dates for submissions to the next two issues of *;login:* are February 3, 1998 and April 7, 1998.

ADVERTISING ACCEPTED

;login: offers an exceptional opportunity to reach 9,000 leading technical professionals worldwide.

Contact: Cynthia Deno
cynthia@usenix.org
408 335 9445

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES